ISSN: 3065-1220 Volume 2, Number 6, December 2025

Hans汉斯

嵌入式技术与智能系统

Embedded Technology and Intelligent Systems

Qian Ru Shi Ji Shu Yu Zhi Neng Xi Tong 2025年12月2卷6期

https://www.hanspub.org/journal/etis

Editorial Board

编委名单

ISSN: 3065-1220

https://www.hanspub.org/journal/etis

Embedded Technology and Intelligent Systems, Vol.2, No.6, 337-390
嵌入式技术与智能系统, 2 卷 6 期
Published Online December 2025 in Hans (https://www.hanspub.org/journal/etis)

TABLE OF CONTENTS
目　　录

期刊信息
期刊中文名称：《嵌入式技术与智能系统》
期刊英文名称：**Embedded Technology and Intelligent Systems**
期刊缩写：**ETIS**
出刊周期：双月刊
语　　种：中文

出版机构：汉斯出版社(Hans Publishers, https://www.hanspub.org/)
编辑单位：《嵌入式技术与智能系统》编辑部
主　　编：何立民，北京航空航天大学教授
网　　址：https://www.hanspub.org/journal/etis

订阅信息
订阅邮箱：sub@hanspub.org
订阅价格：180 美元每年

广告服务
联系邮箱：adv@hanspub.org

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(6), 337-347
Published Online December 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.26033

Hans汉斯

大模型在嵌入式软件开发中的应用

董岩博，蒋立伟，林金龙

北京大学软件与微电子学院，北京

收稿日期：2026年2月2日；录用日期：2026年5月1日；发布日期：2026年5月27日

摘　要

嵌入式软件与硬件和领域知识高度相关，开发难度大，开发周期长。以Transformer架构为核心的大语言模型(LLMs)凭借强大的代码理解与生成能力，为提升嵌入式软件开发效率提供了可能。文章介绍大语言模型在嵌入式软件开发中的应用现状，包括应用过程、方法和常用的大语言模型及其技术特性，并重点梳理大模型在嵌入式软件开发中的应用场景，以及嵌入式软件代码缺陷检测的核心技术路径、场景适配方案。文章通过综述现有研究成果，为嵌入式软件研究和开发人员应用大模型提供参考，助力嵌入式软件开发方法的智能化转型。

关键词

大语言模型，嵌入式软件，代码生成，代码缺陷检测

Applications of Large Language Models in Embedded Software Development

Yanbo Dong, Liwei Jiang, Jinlong Lin

School of Software & Microelectronics, Peking University, Beijing

Received: February 2, 2026; accepted: May 1, 2026; published: May 27, 2026

Abstract

Embedded software is highly correlated with hardware and domain knowledge, featuring high development difficulty and long development cycles. Large Language Models (LLMs) centered on the Transformer architecture, with their robust capabilities in code understanding and generation, have made it possible to improve the efficiency of embedded software development. This paper introduces the current application status of large language models in embedded software development, including the application processes, methods, commonly used large language models and their technical characteristics. It also focuses on sorting out the application scenarios of large models in embedded

文章引用: 董岩博, 蒋立伟, 林金龙. 大模型在嵌入式软件开发中的应用[J]. 嵌入式技术与智能系统, 2025, 2(6): 337-347. DOI: 10.12677/etis.2025.26033

software development, as well as the core technical approaches and scenario adaptation schemes for code defect detection in embedded software. Through a review of existing research findings, this paper provides a reference for embedded software researchers and developers in the application of large models, and facilitates the intelligent transformation of embedded software development methodologies.

Keywords

LLMs, Embedded Software, Code Generation, Code Defect Detection

1. 引言

嵌入式系统已广泛渗透于工业控制、智能终端、汽车电子、物联网等关键领域，其软件开发质量与效率决定着终端产品的性能与竞争力。嵌入式软件开发需要同时兼顾硬件资源约束、实时性要求、底层驱动适配与上层应用逻辑，导致开发周期长、调试难度大，对开发者的综合能力要求较高。随着物联网技术的普及，嵌入式设备数量激增且场景愈发多元，传统依赖人工经验的开发模式已难以应对规模化、定制化的开发需求，亟需新兴技术赋能产业升级。

以 Transformer 架构为核心的大语言模型(LLMs)凭借千亿级参数规模与海量数据预训练优势，可以实现对自然语言与编程语言的深度理解[1]。从通用代码生成、智能调试到需求文档自动化生成，GitHub Copilot 等工具的普及使编码效率大大提升，也验证了大语言模型在代码编写及其他软件开发任务中的巨大潜力。

当前，大语言模型在嵌入式软件开发中的应用仍处于探索阶段，对于这一技术的研究已经取得一定阶段性成果，本文将现有的大语言模型应用于嵌入式开发的研究成果进行整合，对现有的研究方法进行介绍，并对未来的发展方向进行展望，本文将有助于相关领域研究者系统探究大语言模型与嵌入式软件开发的融合路径，明确其适用场景与技术边界，构建高效的人机协同开发框架，为推动嵌入式开发模式智能化转型提供技术参考，助力嵌入式产业在智能化浪潮中实现高质量发展。

2. 代码大语言模型

代码大语言模型(Code LLMs)和通用大语言模型一样，采用 Transformer 架构，其预训练的核心数据是大规模无标签代码语料库，仅搭配少量文本和数学数据，而通用大语言模型的预训练则以大规模文本数据为主，仅融入少量代码和数学数据以提升逻辑推理能力。部分代码大语言模型(如 Qwen2.5-Coder [2])会在训练过程中引入合成数据。

与通用大语言模型类似，代码大语言模型也可分为三类架构：仅编码器型、仅解码器型、编码器-解码器型。仅编码器型通常用于生成代码理解类任务，例如类型预测、代码检索、克隆检测等；仅解码器模型主要用于生成类任务中，例如代码生成、翻译、摘要等；编码器－解码器模型可完成代码理解和生成两类任务，但在特定任务上表现未必优于前两者[1]。

代码大语言模型中 Transformer 的关键模块包括：多头自注意力模块——捕捉语义关系、位置感知前馈网络——优化序列嵌入位置、残差连接与归一化——缓解梯度问题、位置编码——补充位置信息[1]。

代码生成专用大语言模型(LLMs for code generation)指利用大语言模型从自然语言描述生成源代码的技术，这一过程也被称为“自然语言到代码(NL2Code)”任务。输入是自然语言描述，包括编程问题陈述，也可包含编程上下文(如函数签名)，而后模型生成对应源代码，生成的代码再通过编译器/解释器执行单元测试，根据执行反馈(如错误信息)迭代优化先前生成的代码，形成闭环优化流程。

代码生成任务所采用的 Transformer 架构主要分为两类：编码器–解码器架构和仅解码器架构。编码器–解码器架构同时包含编码器和解码器，编码器负责处理输入数据并生成表征，解码器基于这些表征生成输出；仅解码器架构仅保留 Transformer 的解码器部分，通过单一层堆叠同时完成输入处理和输出生成。因此，编码器–解码器架构适用于需要在不同输入和输出域之间建立映射的任务，而仅解码器架构则专门针对序列生成和续写类任务设计。

3. 大模型嵌入式软件开发模式

3.1. 可行性探究

对于大模型在嵌入式开发中应用的可行性探究工作，涉及开发全流程中的诸多领域。大模型具有极强的代码生成理解能力，因此较早实现验证的便是大模型辅助代码生成的可行性，并且为了完成对大模型生成的嵌入式代码的功能评估，诸多的测试和评估方法也应运而生；一些研究团队通过对大模型在嵌入式系统中的能力与局限的探究，发现大模型的跨软硬件的推理能力也是其核心优势之一，其在嵌入式开发中的价值不仅局限于代码生成，这一优势也使得大模型在更多开发环节中的应用部署具备可能，为全开发流程自动化提供可能[3]。资深固件工程师 Mark 分享的 AI 辅助嵌入式开发实操指南，通过精准配置 AI 工具，全程聚焦 STM32、ESP32 等主流硬件的实操需求，将大模型适配嵌入式场景，覆盖到代码生成、调试、优化、测试全流程，能够解决嵌入式开发中重复编写外设初始化代码、寄存器配置查询繁琐、硬件 bug(如 I2C 时序、SPI 间歇性故障)排查耗时、新手频繁出现同质化问题的痛点，并且可以将嵌入式系统程序调试时间减少 60%，同时提升代码可靠性，完美验证了 LLMs 辅助嵌入式开发的可行性和高效性[4]。Englhardt 等人通过构建首个开源可扩展的硬件在环评估框架，将 LLM 生成的代码直接上传到微控制器(如 Arduino Uno、nRF52832)，通过“传感器–执行器对”与人类编写的基准代码进行物理输出对比，验证功能正确性。基于该框架完成 450 次真实硬件实验，覆盖光敏电阻、超声波测距仪、6 轴 IMU 等典型嵌入式场景，针对 GPT-3.5、GPT-4、PaLM 2 三个主流模型，从硬件理解、代码生成、系统设计三个维度展开深度测试，发现跨域知识与代码生成能力 GPT-4 表现最优，在部分任务中单次提示即可生成完全正确的代码，在 50 次实验中 66%能生成可用的 I2C 接口，还能编写寄存器级驱动、LoRa 通信代码；针对 nRF52832 的电源优化建议，使电流消耗降低 740 倍至 12.2 μA (接近人类专家优化的 8.6 μA)。硬件规格理解方面，GPT-4 能精准识别常见芯片(如 BME280、LSM6DSO)的 I2C 地址、寄存器功能、通信协议，甚至能解释 datasheet 中的位字段含义；对虚构传感器的寄存器表，可生成符合规格的驱动伪代码。并且即使代码不完全正确，LLM 仍能提供 actionable 建议，如检查接线、修正 I2C 地址、优化采样率等，尤其对硬件新手帮助显著[3]。

3.2. 框架概述

当前大模型在嵌入式软件开发中的应用主要围绕在代码生成与调试阶段，通过构建实现“需求描述–生成代码–编译验证–问题优化”的闭环框架来完成开发任务。Englhardt 等人提出的人机协同开发模式[3]，将大模型作为辅助工具，通过结构化工作流整合人类领域知识与大模型的代码生成能力，覆盖需求分析、代码生成、调试部署全流程：(1) 首先要求用户提供完整上下文，如硬件型号、引脚连接、库版本、预期行为，避免模糊表述；(2) 将编译器错误信息反馈给 LLM；(3) 详细描述物理行为异常，利用

LLM 可关联硬件逻辑提供解决方案的能力；此外，还可以考虑拆分复杂任务，在模块化生成函数后再进行人工整合，还可以提供示例代码约束输出、及时更新代码上下文，避免 LLM 陷入数值计算错误。该团队通过使用该方法，使得任务成功率显著提升，LoRa 环境传感器的开发成功率从 25%提升至 100%，零硬件/C++经验的用户可在 40 分钟内完成功能完整的收发器。新手借助 LLM 填补硬件知识缺口，专家通过 LLM 生成模板代码提升效率。经 NASA TLX 评分得到，参与者的任务感知负载从 4.67 降至 3.25，尤其减少了“硬件接线排查”“寄存器配置”等繁琐工作的负担[3]。

3.3. 跨平台嵌入式开发

跨平台嵌入式开发也是当前研究方向之一，应用大模型将一个平台的代码转化成适配另一个平台的代码并实现相同功能，用于节省跨平台开发需要从零做起的高时间成本。Xu 等人提出的方法通过外部知识检索补充大模型的硬件知识，利用编译器反馈修正语法错误，提升代码的跨平台适配性与正确性[5]。首先利用语言大模型检索增强生成，提取同类硬件的预生成代码作为参考，辅助大模型复用成熟 API 与逻辑，避免重复错误；再利用推理大模型实现编译器反馈优化，捕获框架的语法错误信息，反馈给大模型进行针对性修正，降低跨平台迁移的语法适配成本；最终基于硬件引脚映射、编程语言差异(如 Arduino →MicroPython)自动调整代码和电路连接方案。

4. 大模型代码生成

代码生成是大模型辅助嵌入式开发的核心场景，开发者可以使用精准的自然语言指令部署大模型使其实现嵌入式代码的生成。对于初步生成的代码所存在的部分语法、编译问题和功能缺失，还需要实现反馈优化，通过向大模型反馈其输出结果在运行中存在的问题，使其重新思考理解并生成优化代码，增强其生成能力，完成设计任务[3] [5]。并且不同 AI 工具有着不同的嵌入式专属配置，包括提示词设计、上下文配置、嵌入式训练方法(包括运行软件配置、禁用文件提示等)、大模型思考模式等等，不同的模型配置、设计框架、优化方法也会使得生成结果在实际应用测试中表现出较大差异。下面介绍面向不同应用场景的不同设计和优化方法，以及所使用的大语言模型所达到的生成效果。

4.1. 提示词设计

首先，通过系统提示词定义 LLM 的目标角色、核心能力和行为准则；然后，通过用户提示词明确 LLM 的具体任务——根据给定规格完成函数定义。并且在提示词中可以提供 C 程序的结构框架，包括头文件、全局变量和函数头等信息。

4.1.1. 语言规格

针对具体任务描述，从语言规格角度可以分为高层自然语言规格(HLNL)、低层自然语言规格(LLNL)和 ACSL 规格[6]。HLNL 聚焦系统需实现的核心功能与约束，明确系统要达成什么目标，如“若存在刹车灯请求，则应激活卡车车灯”；LLNL 目标是实现高层自然语言规格，明确系统如何实现目标的具体要求，如“若运行状态为正常运行或有限功能紧急停机，且供电电压未处于低电平，则应启用刹车灯”；ACSL 规格是以 ANSI/ISOC 规格说明语言(ACSL)编写的形式化规格说明集。

Patil M S 等人提出的 Spec2code 框架利用含前置条件、后置条件、帧条件的 ACSL 形式化规范与高、低层级自然语言规范来覆盖功能与约束定义，生成初始代码，再通过将编译器、验证工具等“批评者”提供的反馈用于提示词迭代，进行迭代优化与模型微调，持续优化代码。在使用 GPT-3.5 与 GPT-4-turbo，基于零样本链式思考，完成对三款真实汽车控制模块(转向油位检测、刹车灯激活、动力转向备份)的设计实现后，Patil 等人验证了 Spec2code 框架极简版本的可行性，证明无需复杂迭代也能生成可编译、可验

证的工业级代码也说明了技术细节明确的描述规格能提高所生成代码的可用性[6]。

4.1.2. 连接约束

除了给定任务描述外，还可考虑为大模型提供电路图，规定连接要求。也可以要求大模型实现跨平台设计，即要求大模型重新设计代码，将某一硬件平台的代码和原理图迁移至另一平台，实现相同功能。EmbedAgent 是一个嵌入式代码生成和评估方法[5]，其 EmbedBench 基准是三个核心任务场景。三个场景分别是：(1) 给定任务描述和电路原理图，生成嵌入式代码；(2) 仅给定任务描述，自主设计电路原理图并编写对应代码；(3) 给定某硬件平台代码和原理图，要求迁移至另一平台，设计实现代码。

设计结构化电路描述格式(.json)，将硬件连接转化为 LLM 可理解的“组件 ID + 引脚”配对形式，基于 Wokwi 虚拟电路仿真平台，实时监控虚拟硬件的运行状态(而非依赖串行输出)，直接验证生成代码是否符合任务要求[5]。

EmbedBench 基准包含 126 个手动构建的测试案例，覆盖 9 类电子元件，包括 LED、RGB LED、按键、7 段数码管、滑动变阻器等。每个案例含任务描述、参考解决方案和自动化验证用例，确保评估的全面性。Ruiyang Xu 等人在对 10 个主流 LLM 进行系统性测试后发现在给定原理图时，最优模型 DeepSeek-R1 的 pass@1 仅 55.6%；不指定原理图时降至 50.0%；跨平台迁移任务难度最高，ESP32 平台的 ESP-IDF 框架迁移最优 pass@1 仅 29.4% (Claude 3.7 Sonnet)，而 Raspberry Pi Pico 的 MicroPython 迁移表现相对较好(最优 73.8%)。DeepSeek-R1、Claude 3.7 Sonnet 等推理型 LLM 整体优于聊天型 LLM，但存在像为 7 段数码管生成二进制码时陷入冗余计算等“过度思考”的问题；DeepSeek-V3、Llama-3.3-Instruct 等聊天型 LLM 难以灵活适配任务场景，如无法将预训练的 7 段数码管编码表调整为共阳极配置；DeepSeek-R1-Distill 等 SFT 蒸馏型 LLM 稳定性差，面对陌生任务时性能甚至低于基础模型[5]。QWQ 等推理型 LLM 在自主设计电路时，会选择比参考方案更合理的引脚连接方式，导致其在不提供电路图时的性能更优，说明僵化的预设原理图可能限制 LLM 的推理能力。

4.1.3. 其他优化方法

针对无线通信的功耗、调制方案等专项需求，Medaranga 等人提出通过精准提示引导大模型生成反向散射发射器等无线模块代码，确保通信协议合规性与信号稳定性的方法[7]。主要技术包含：(1) 专项参数明确：即在提示中指定调制方案、中频、比特率、硬件平台等关键参数，避免大模型幻觉；(2) 硬件特性适配：结合反向散射通信的低功耗原理，生成 GPIO 引脚精准控制逻辑，确保信号调制和传输符合无线标准。

针对传统手动方法的低效问题和现有 LLM 在领域适配性上的不足，Lin 等人提出一种“监测–分析–规划–执行–知识”(MAPE-K)架构，设计面向航空嵌入式系统的 LLM 结构化流程[8]。通过“分类–形式化–组装”三步法 + BNF 语法约束，解决 LLM 的领域适配性问题。以真实子系统为案例，对比不同参数 LLM、手动方法、直接 LLM 方法，量化证明了所提方法在效率和准确性上的优势，具备实际工程应用价值。主要技术包含(1) 领域知识建模：将航空系统需求划分为监测、决策、通信等 5 大领域，定义 11 类核心操作，明确其语法规则与硬件交互逻辑；(2) 需求规范化：采用巴科斯–诺尔范式(BNF)统一需求表达，包含时间约束、知识引用等关键元素，消除自然语言歧义；(3) 设备信息提取、功能规范形式化(分类、形式化、组装)、约束提取，以确保代码与硬件交互、任务逻辑一致[8]。

此外，基于零样本思维链提示技术设计提示词，可以引导模型在生成最终答案之前，先逐步推导出一系列的中间推理步骤，从而提升大语言模型在复杂推理任务上的表现[6]。

在编译器反馈阶段，针对平台的语法错误问题，将编译器的错误信息反馈给 LLM，弥补大模型在硬件知识、跨平台适配等方面的不足，通过人类参与、工具反馈、知识检索等方式引导其进行修正，实现

优化。EmbedAgent 基于这种方法仅需 1 轮迭代，便可实现语法错误率从 34.1%降至 3.2%，ESP32 迁移任务的 pass@1 从 21.4%提升至 27.8% [5]。

4.2. 生成代码评估方法

对于编译成功的代码，可以从以下角度进行评估：功能正确性、程序等价性和代码质量。功能正确性旨在验证程序的正确性，即是否满足规格说明；程序等价性则是利用已有的代码库中人工编写的代码和所生成的代码进行对比，捕捉二者的语义差异，检验其相对正确性；代码质量可以通过代码行数或其他特定的规则如函数规模、循环要求等进行考量。Patil 等人在 Spec2code 框架中便是使用 Frama-C 中的最弱前置条件演算插件(WP)，基于 ACSL 规格验证程序的功能正确性，采用 Z3 和 Alt-Ergo 作为后端求解器；通过 diffkemp 工具验证功能正确性，该工具可对 LLVM 字节码形式的程序进行指令级等价性验证，验证过程采用一组语义保留转换技术；采用定量与定性相结合的方式评估代码质量，除统计对比代码行数外，还人工检查其对“十大规则”的符合程度[6]。

4.3. 全自动化平台构建

对于整个开发工作，目前有研究致力于实现构建端到端自动化嵌入式开发全流程。其中 EmbedGenius 就是一个面向通用嵌入式 IoT 系统的全自动化开发框架，包含端到端自动化嵌入式开发全流程，解决硬件依赖解析、库知识缺失、编程复杂性三大痛点，无需人工干预即可完成从需求到部署的闭环[9]。核心技术包括(1) 硬件依赖解析：基于名称匹配、版本迭代频率、架构兼容性评分，自动选择最优硬件库，适配多模块组合场景；(2) 库知识注入：从库的头文件与示例中提取 API 声明、参数、使用顺序，构建 API 表与组件功能表，注入大模型内存；(3) 嵌套反馈循环：通过“编译循环”(确保语法正确)与“闪存循环”(验证功能正确性)的嵌套反馈，结合 DEBUG 信息实现错误自修正；(4) 安全防护：对高风险操作进行开发者确认，对敏感信息进行占位符替换。Yang 等人通过“大规模基准测试 + 实际案例验证”，全面验证 EmbedGenius 的性能，包括 4 个主流开发平台(Uno R3、NUCLEO-L4、Nano RP2040、Nano ESP32)、测试四种大模型性能(GPT-4o、GPT-4、Claude-3.5、GPT-3.5)、使用 71 个硬件模块(传感器、通信模块、显示器等)、355 个 IoT 任务(分 3 个难度等级，覆盖环境监测、运动控制、数据通信等场景)。以 3 个需人类参与的开发方法(LLM-Prompt、Duinocode、LLM-direct)作为对比，从编码准确率(API 调用与参数配置的正确性)、任务成功率(一次执行完成所有子功能的概率)两个角度评估其性能。最终得出 GPT-4o 综合性能、延迟、成本最优，支撑系统达到 95.7%编码准确率、86.5%任务成功率的核心指标；GPT-4/Claude-3.5 在更强的长文本解析(如复杂库文件)或稳定性需求下可替换使用，但需接受更高延迟/成本；而 GPT-3.5 则需避免用于复杂场景：仅推荐用于“单传感器读取”“LED 控制”等简单任务，或对成本极度敏感的批量开发场景。论文还通过两个真实场景案例，验证系统的实用性：环境监测系统：整合 DHT11 传感器、LoRa 模块、OLED 显示器，2.6 分钟完成开发，部署于森林持续监测温度，支持多 LLM (GPT-4o、Claude-3.5 等)，成功率 ≥80%；远程控制系统：整合 PIR 人体传感器、继电器、Wi-Fi 模块，3.1 分钟完成开发，部署于智能家居实现风扇自动控制，自动生成实时更新的 Web 控制界面，编码准确率 ≥90% [9]。

EmbedAgent 也构建出一个端到端自动化流水线，通过开发自动化提交机器人，将 LLM 生成的原理图和代码映射到 Wokwi 的虚拟环境中，自动执行测试用例并返回结果，无需手动组装硬件，便可实现高效、可扩展的评估[5]。

5. 大模型生成的嵌入式代码的检测方法

嵌入式系统广泛应用于工业控制、汽车电子等关键领域，代码安全性直接决定设备可靠性，嵌入式

系统中约 73%的安全漏洞源于代码缺陷[10]，嵌入式领域安全漏洞中 32%源于缓冲区溢出，25%涉及内存泄漏[10]。根据 NIST SP 800-53 的安全控制框架，这些漏洞类型对应访问控制、边界保护等安全控制族[11]。嵌入式系统的硬件相关性(如寄存器操作、中断机制)与资源约束(内存、算力有限)，使得大模型生成的代码易存在隐性缺陷与显性错误。

5.1. 嵌入式代码缺陷检测

嵌入式代码缺陷特指因硬件交互逻辑不当、资源分配不合理或规格实现偏差导致的潜在问题，如中断竞态条件、寄存器配置错误、缓冲区溢出等，其检测需深度关联嵌入式领域特性与代码语义。本节分析数据集增强与 RAG 检索增强两类检测技术的进展。

5.1.1. 数据集增强与领域适配路径

通用代码数据集难以覆盖嵌入式系统的硬件交互特性，如 MCU 寄存器操作、汇编/C 混合编程，数据集的领域适配与规模扩充成为提升缺陷检测精度的基础。Bhandari 等人提出的 IoTvulCode 框架[12]是该路径的代表性研究，其核心思路是构建面向 IoT 嵌入式系统的大规模标注数据集，通过 FlawFinder、CppCheck、Rats 三款静态分析工具联合扫描 Linux-rpi、FreeRTOS 等 11 类主流 IoT 项目的源代码，最终形成包含 1,014,548 条语句(65,135 条漏洞样本)和 548,089 个函数的高质量数据集，标注遵循 CWE 标准，支持二进制分类(漏洞/无漏洞)与多分类(具体漏洞类型)任务。在模型设计上，该框架采用 RNN、LSTM 等序列模型，针对嵌入式代码短语句(平均 38 字符)的特性优化输入长度与词嵌入策略，最终实现语句级二分类准确率 99.1%、召回率 88.5%，多分类任务中准确率、精确率与召回率均达 99%，显著优于 iDetect 等依赖小数据集的传统工具，其数据集规模较 iDetect 扩大 162.4 倍，有效缓解了嵌入式领域标注数据稀缺的问题。

该路径的创新延伸集中在数据集场景化优化与模型轻量化适配。Tang 等人在 ICFEM 2023 的研究[13]基于 IoTvulCode 的数据集构建思路，筛选 ESP32、Raspberry Pi 等物联网硬件的专属代码样本，对 CodeLlama-7B 进行领域微调，通过归一化不同厂商 MCU 的宏定义与寄存器命名，减少代码变体对模型的干扰，最终在 IoT-Security-DataSet 上实现漏洞检测 F1 值 81.7%，较通用 CodeLlama 提升 25%。此外，Hanif 等人的 VulBERTa 模型[14]针对嵌入式 C/C++代码的语法特性优化分词策略，保留寄存器命名、宏定义的完整性，在 IoTvulCode 数据集上的缺陷检测 F1 值达 89.2%，较通用 BERT 模型提升 17%，验证了数据集与模型结构协同优化的有效性。

5.1.2. RAG 增强的多阶段静态检测路径

嵌入式代码缺陷常隐藏于规格说明与代码实现的语义偏差中，传统静态分析难以关联非结构化规格文档(如硬件手册、协议规范)，RAG (Retrieval Augmented Generation)技术通过检索增强实现规格与代码的语义对齐，成为解决该问题的核心路径。Qayyum 等人在 ACM TDES 上提出的 3 阶段 4 步骤方法[15]是该路径的典型代表，其技术核心在于将 RAG 与迭代信息补充相结合：首先通过预处理将规格文档拆分为 500 token 的片段并嵌入向量库，再对目标 Verilog 代码逐行进行语义检索，获取相关规格片段作为 LLM 上下文；若初始检测失败，则进入 4 阶段 bug closure 流程——逐步增大规格文档检索块至 1000 token、补充前后各 5 行相邻代码、将复杂的硬件逻辑表达式扁平化拆解为原子级子表达式、提取功能属性，通过多维度信息补充缩小缺陷定位范围。该方法针对 AES、I2C、USB 等 5 类 OpenTitan IP 的 9 类缺陷(含常量错误、切片错误、敏感列表错误等)进行测试，最终缺陷修复率达 60%，另有 11%的缺陷被准确识别但未完成修复，成功解决了传统 RAG 对常量类缺陷检测失效的问题[16]。

类似地，Qayyum 等人在 LAD 2024 的研究[16]进一步简化该路径，通过 RAG 检索规格文档与代码

上下文，针对 AON 定时器、UART、EDN 三类 OpenTitan IP 的功能缺陷，实现除常量错误外 100%的修复率，验证了该路径在专用嵌入式 IP 检测中的适配性。该路径的创新延伸主要体现在检索维度的拓展与硬件知识的深度融合：Englhardt 等人在 arXiv 2023 的研究[3]通过 RAG 补充 MCU 硬件手册、寄存器配置表等领域知识，评估 GPT-4、CodeLlama 等模型对嵌入式特有缺陷的检测能力，发现增强后模型对中断处理漏洞、寄存器配置错误的检测准确率从 78.3%提升至 89.1%，硬件交互类缺陷漏检率从 42%降至 28%。Tang 等人的研究[13]则将 RAG 检索范围扩展至 IoT 设备的通信协议文档(如 SPI、I2C 协议规范)，使模型能精准识别协议交互中的逻辑缺陷，在工业物联网网关代码检测中，协议相关缺陷的识别准确率较通用 RAG 提升 32%，进一步拓宽了该路径的应用场景。

5.2. 嵌入式代码 bug 检测

嵌入式代码 bug 特指导致系统运行异常的显性错误，如语法错误、内存访问错误、编译错误等，其检测需兼顾实时性与修复精准性，同时适配嵌入式系统的资源约束。本节讨论自反思迭代修复与数据集优化方法的技术进展与工程实践效果。

5.2.1. 自反思驱动的迭代修复路径

大模型生成的嵌入式代码 bug 常伴随编译器反馈缺失或上下文信息不足的问题。自反思机制通过引入编译器错误信息、迭代优化修复方案，成为提升 bug 检测与修复效率的关键路径。Cui 等人提出的 OriGen 框架[17]是该路径的标杆之作，其技术创新体现在两个核心层面：一是代码到代码的数据增强策略，通过 Claude3-Haiku 对开源 RTL 代码进行功能描述生成与重构，过滤超过 300 行或非标准格式的低质量样本，同时收集编译失败案例构建包含“错误代码–编译器反馈–修复代码”的错误修复数据集；二是双 LoRA 模型架构与自反思循环，Gen LoRA 负责基于自然语言指令生成初始 RTL 代码，Fix LoRA 在编译器反馈触发下，分析语法错误、模块接口错误等信息并迭代修复代码，直至通过 Icarus Verilog 编译器验证。该方法在 VerilogEval-Human 基准上 pass@1 达 54.4%，超越 GPT-4 Turbo，在 VerilogFixEval 基准(编译错误修复专项)中，自反思修复能力较 GPT-4 提升 19.9%，成功解决了开源 LLM 在 RTL 代码语法错误、寄存器传参错误修复上的短板。

该路径的创新延伸聚焦于反馈机制优化与端侧部署适配。Tsai 等人提出的 RTLFixer 框架[18]将自反思与 ReAct 策略结合，通过 RAG 检索人类专家修复案例库，为 LLM 提供针对性修复指导，针对 RTL 语法错误的修复成功率达 83%，较无指导自反思提升 27%。Thakur 等人的 AutoChip 框架[19]则直接将编译器错误信息输入 LLM，无需检索外部数据库，通过多轮对话引导模型定位 bug 根源，在 8 位累加器微处理器设计中，编译错误修复效率较传统方法提升 4 倍，且修复代码的硬件资源占用率降低 12%。针对嵌入式端侧部署需求，OriGen 通过模型量化与剪枝技术，将 Fix LoRA 模块的推理耗时控制在 200 ms/千行代码以内，可直接部署于 ARMCortex-M4/M7 系列 MCU，为端侧实时 bug 检测与修复提供了可行方案。

5.2.2. 数据集优化的批量 bug 检测路径

嵌入式代码 bug 的多样性，如 MCU 架构差异、编程语言特性等要求检测模型具备强泛化能力，而高质量大规模数据集是实现这一目标的核心支撑。Bhandari 等人的 IoTvulCode 框架[12]在 bug 检测领域的拓展应用，充分体现了数据集优化的价值。该框架构建的数据集覆盖 11 类主流 IoT 项目，包含 65,135 条 bug 样本(涵盖缓冲区溢出、整数溢出、空指针等 10 类主要 CWE 类型)，通过 srcML 工具提取函数级与语句级代码片段，结合 Guesslang 进行编程语言分类，为批量 bug 检测提供了标准化数据支撑。在模型层面，采用 RNN、LSTM 等序列模型，针对嵌入式代码短语句、高复用性的特点优化输入长度(语句级 150 token、函数级 1024 token)与词嵌入策略，最终实现语句级 bug 检测二分类准确率 99.1%、召回率 88.5%，多分类准确

率 99%，较 iDetect 等传统工具的损失值降低 78%，展现出大规模数据集对模型泛化能力的提升作用。

该路径的创新主要体现在数据集场景化细化与模型适配优化。Englhardt 等人在 arXiv2023 的研究[3]通过数据集扩充，加入 STM32、Arduino 等平台的运行时 bug 样本(如定时器中断冲突、SPI 通信时序错误)，使 GPT-4 对这类嵌入式特有 bug 的检测准确率从 62%提升至 79%，验证了数据集场景化优化的重要性。此外，针对嵌入式系统的实时性要求，部分研究通过数据集筛选，保留短代码片段(≤50 行)的 bug 样本，训练轻量化模型，使检测耗时控制在 10 ms/行以内，满足嵌入式开发的实时检测需求。

5.3. 大模型检测技术的场景适配性分析

不同嵌入式应用领域的硬件特性、功能需求与安全标准存在显著差异，对大模型检测技术的适配性提出了差异化要求。为直观呈现各场景下的最优检测策略及性能表现，我们用下表从应用领域、核心约束、典型漏洞类型、推荐检测方案及代表成果与性能五个维度，系统梳理大模型检测技术的场景适配情况。

Table 1. Scenario adaptability analysis of large model detection technology
表 1. 大模型检测技术的场景适配性分析

应用领域	核心约束	典型漏洞类型	推荐检测方案	代表成果与性能
工业控制	高可靠性、实时检测、多架构适配	硬件交互类、内存溢出	EmbedVulDet 混合模型 + 边缘计算节点	F1 值 0.94，误报率 < 5%，检测耗时 < 200 ms/段
汽车电子	功能安全合规(ISO 26262)、漏洞可复现	实时性冲突、CAN 总线协议漏洞	TraceBERT + 动态轨迹分析	漏洞根源定位准确率 89%，支持 ISO 26262 报告生成
消费电子(MCU)	低算力、低功耗、轻量化部署	控制流异常、简单内存漏洞	量化压缩 CodeBERT (4-bit) + 静态分析	模型体积 < 100 MB，准确率 87.5%，支持本地检测
航空航天	极高安全性、全链路溯源、零误判	复杂逻辑漏洞、冗余代码引发的性能问题	多模态大模型 + 人工复核闭环	准确率 99.2%，支持漏洞关联影响分析

5.4. 现存核心问题与技术挑战

5.4.1. 技术层面的核心瓶颈

1) 开源数据集质量与规模不足：嵌入式领域高质量标注数据稀缺，IoTvulCode 数据集虽达百万级样本，但小众硬件相关样本占比不足 5% [12]。

2) 跨架构适配能力有限：现有模型在 ARM 与 RISC-V 架构间迁移检测准确率下降 25%~30%，难以适配异构硬件[12]。

3) 编译器反馈利用率低：开源模型对编译错误信息的解析能力薄弱，OriGen 的 self-reflection 机制虽有改善，但复杂错误修复率仍低于 40% [14]。

4) 功能规格书依赖性强，模糊或缺失规格会导致准确率下降 15%~20% [3]。

5.4.2. 未来研究方向与突破路径

1) 硬件感知的专用大模型构建：通过“通用代码预训练 - 硬件知识微调 - 场景数据精调”三级框架，提升异构硬件适配能力[12]。

2) 小样本与零样本检测技术创新：基于 Prompt Tuning 等方法，通过漏洞模式模板 + 少量标注样本实现高效检测，50 条数据场景已验证可行性[13]。

3) 轻量化与边缘部署优化：采用剪枝、量化等技术，让模型体积减少，检测耗时降低，准确率提升

[12]。

4) 工程化工具链集成：开发与 Keil、IAR 等环境集成的插件，实现“编码－编译－检测－修复”无缝衔接[14]。

6.展望

大模型在嵌入式软件开发中的应用已经开始展现巨大潜力，在代码生成、驱动开发与缺陷检测等核心环节，有效缓解了硬件适配复杂、专业知识依赖度高的传统痛点。但当前技术仍存在不少关键问题，未来可关注三个核心的方向来深化探索。

在技术适配层面，应重点突破跨架构与硬件感知的短板。现有模型在 ARM 与 RISC-V 等异构硬件间迁移时性能下滑明显，对小众硬件的支持不足，后续可以通过“通用预训练－硬件知识微调－场景精调”模式，补充寄存器配置、通信协议等专属知识，结合小样本学习降低标注数据依赖。同时优化编译器反馈解析机制，提升模型对复杂编译错误的修正能力，减少开发中的语法与逻辑漏洞。

在场景与工具层面，推进垂直领域定制与全流程整合。汽车电子、工业控制等领域对合规性、实时性要求差别很大，需要做针对性优化：汽车电子重点关注 ISO 26262 合规性检测，工业控制主要提升硬件交互缺陷识别精度，消费电子则通过剪枝、量化技术优化轻量化部署。工具链方面，推动大模型与 Keil、IAR 等开发环境集成，结合虚拟仿真平台构建“编码－检测－修复”自动化流程，优化人机协同模式，让开发者聚焦核心逻辑，模型承担重复编码与基础调试。

安全保障层面，完善漏洞防控体系。嵌入式代码直接影响设备可靠性，需建立更全面的漏洞分类标准，提升对内存溢出、实时性冲突等核心问题的“检测－修复”效率，避免大模型生成代码引入隐蔽风险，确保系统运行稳定。

总体而言，大模型对嵌入式软件开发有提升效率、整合流程的巨大潜力，但跨架构适配性不足、工具链无法衔接等问题仍需解决。未来研究需紧扣实际开发流程中的问题，通过技术优化、工具整合与安全防护，推动大模型在嵌入式软件开发各环节的深度参与并发挥重大作用。

参考文献

[1] Jiang, J., Wang, F., Shen, J., *et al*. (2024) A Survey on Large Language Models for Code Generation. https://dl.acm.org/doi/full/10.1145/3747588

[2] Hui, B., Yang, J., Cui, Z., *et al*. (2024) Qwen2.5-Coder Technical Report. https://arxiv.org/abs/2409.12186

[3] Englhardt, Z., Li, R., Nissanka, D., Zhang, Z., Narayanswamy, G., Breda, J., *et al*. (2024) Exploring and Characterizing Large Language Models for Embedded System Development and Debugging. *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, Honolulu, 11-16 May 2024, 1-9. https://doi.org/10.1145/3613905.3650764

[4] Stop Wasting Hours Debugging: Use AI to Write Better Embedded Systems Code. https://markaicode.com/ai-embedded-systems-code/

[5] Xu, R., Cao, J., Wu, M., *et al*. (2025) Embed Agent: Benchmarking Large Language Models in Embedded System Development. https://arxiv.org/abs/2506.11003

[6] Patil, M.S., Ung, G. and Nyberg, M. (2024) Towards Specification-Driven LLM-Based Generation of Embedded Automotive Software. In*: Lecture Notes in Computer Science*, Springer, 125-144. https://doi.org/10.1007/978-3-031-75434-0_9

[7] Medaranga, P., Shah, D., Kandala, S.V. and Varshney, A. (2024) POSTER: Simplifying the Networking of Wireless Embedded Systems Using a Large Language Model. *Proceedings of the ACM SIGCOMM* 2024 *Conference*: *Posters and Demos*, Sydney, 4-8 August 2024, 78-80. https://doi.org/10.1145/3672202.3673752

[8] Lin, J., Luo, Y., Chen, X., Gu, B. and Jin, Z. (2025) Automatic Generation of Structured Requirements for Aerospace Embedded Systems Using LLMs. 2025 *IEEE* 33*rd International Requirements Engineering Conference Workshops* (*REW*), Valencia, 1-5 September 2025, 181-188. https://doi.org/10.1109/rew66121.2025.00028

[9] Yang, H., Li, M., Han, M., *et al*. (2024) Embed Genius: Towards Automated Software Development for Generic

Embedded IoT Systems. https://arxiv.org/abs/2412.09058

[10] MITRE Corporation and SANS Institute (2024) 2024 CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html

[11] National Institute of Standards and Technology (2022) Security and Privacy Controls for Information Systems and Organizations (NIST SP 800-53 Revision 5). https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final

[12] Bhandari, G.P., Assres, G., Gavric, N., Shalaginov, A. and Grønli, T. (2024) IoTvulCode: AI-Enabled Vulnerability Detection in Software Products Designed for IoT Applications. *International Journal of Information Security*, **23**, 2677-2690. https://doi.org/10.1007/s10207-024-00848-6

[13] Yang, Y.L. (2023) IoT Software Vulnerability Detection Techniques through Large Language Model. In: *Lecture Notes in Computer Science*, Springer, 285-290. https://doi.org/10.1007/978-981-99-7584-6_21

[14] Hanif, H. and Maffeis, S. (2022) VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection. 2022 *International Joint Conference on Neural Networks* (*IJCNN*), Padua, 18-23 July 2022, 1-8. https://doi.org/10.1109/ijcnn55064.2022.9892280

[15] Qayyum, K., Jha, C.K., Ahmadi-Pour, S., Hassan, M. and Drechsler, R. (2025) LLM-Assisted Bug Identification and Correction for Verilog HDL. *ACM Transactions on Design Automation of Electronic Systems*, **30**, 1-28. https://doi.org/10.1145/3733237

[16] Qayyum, K., Hassan, M., Ahmadi-Pour, S., Jha, C.K. and Drechsler, R. (2024) From Bugs to Fixes: HDL Bug Identification and Patching Using LLMs and Rag. 2024 *IEEE LLM Aided Design Workshop* (*LAD*), San Jose, 28-29 June 2024, 1-5. https://doi.org/10.1109/lad62341.2024.10691874

[17] Cui, F., Yin, C., Zhou, K., Xiao, Y., Sun, G., Xu, Q., *et al*. (2024) OriGen: Enhancing RTL Code Generation with Code-to-Code Augmentation and Self-Reflection. *Proceedings of the* 43*rd IEEE/ACM International Conference on Computer-Aided Design*, New York, 27-31 October 2024, 1-9. https://doi.org/10.1145/3676536.3676830

[18] Tsai, Y.D., Liu, M. and Ren, H. (2024) RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Model. *Proceedings of the* 61*st ACM/IEEE Design Automation Conference*, San Francisco, 23-27 June 2024, 1-6. https://doi.org/10.1145/3649329.3657353

[19] Thakur, S., Blocklove, J., Pearce, H., *et al*. (2023) AutoChip: Automating HDL Generation Using LLM Feedback. https://arxiv.org/abs/2311.04887

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(6), 348-354
Published Online December 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.26034

Hans汉斯

面向AI融合的嵌入式系统课程改革探索

毕　盛，董　敏，冼　进，汪秀敏

华南理工大学计算机科学与工程学院，广东 广州

收稿日期：2026年2月2日；录用日期：2026年5月1日；发布日期：2026年5月27日

摘　要

面对人工智能与嵌入式系统深度融合的产业趋势，以及新工科建设对跨学科复合型人才的培养要求，传统课程体系存在“知识断层”。研究系统阐述了《智能嵌入式技术》课程改革与实践。课程构建了涵盖“系统认知、硬件平台、软件栈、算法部署、应用集成”五个层次的全栈知识体系，旨在打通从异构硬件到智能应用的知识链路。在教学实施上，创新采用“理论－实践－竞赛”三维驱动模式：通过模块化精讲与案例剖析夯实基础；以贯穿学期、覆盖“感知－决策－控制”闭环的综合性项目驱动深度工程实践；并通过对接高水平学科竞赛，淬炼学生创新与解决复杂工程问题的能力。

关键词

嵌入式系统，人工智能，智能嵌入式，新工科，教学改革

Exploration of Embedded Systems Curriculum Reform for AI Integration

Sheng Bi, Min Dong, Jin Xian, Xiumin Wang

School of Computer Science & Engineering, South China University of Technology, Guangzhou Guangdong

Received: February 2, 2026; accepted: May 1, 2026; published: May 27, 2026

Abstract

Faced with the industrial trend of deep integration between artificial intelligence and embedded systems, as well as the requirements of Emerging Engineering Education for cultivating interdisciplinary and composite talents, traditional curriculum systems suffer from a “knowledge gap.” This study systematically elaborates on the reform and practice of the “Intelligent Embedded Technology” course. The course constructs a full-stack knowledge system covering five levels: “system cognition, hardware platform, software stack, algorithm deployment, and application integration,” aiming to bridge the knowledge pathway from heterogeneous hardware to intelligent applications. In terms of teaching

文章引用: 毕盛, 董敏, 冼进, 汪秀敏. 面向 AI 融合的嵌入式系统课程改革探索[J]. 嵌入式技术与智能系统, 2025, 2(6): 348-354. DOI: 10.12677/etis.2025.26034

implementation, it innovatively adopts a "theory-practice-competition" three-dimensional driven model: solidifying the foundation through modular intensive lectures and case analysis; driving in-depth engineering practice with semester-long comprehensive projects covering the "perception-decision-control" closed loop; and refining students' innovation and complex problem-solving abilities by connecting with high-level academic competitions.

Keywords

Embedded Systems, Artificial Intelligence, Intelligent Embedded Systems, Emerging Engineering Education, Teaching Reform

1. 引言

随着人工智能、物联网和机器人等技术的迅猛发展，嵌入式系统正经历着从传统控制到智能感知、决策与协同的根本性变革。智能嵌入式系统作为连接数字世界与物理世界的桥梁，不仅继承了嵌入式设备低功耗、小型化、高可靠性的特点，还在有限资源条件下承担起深度学习推理、多模态感知与实时控制等复杂任务。这一趋势对人才培养提出了新的要求——既需要掌握嵌入式系统硬件与软件的基础知识，又要具备人工智能模型设计、优化与部署的能力。

为应对这一技术融合与产业需求，我校计算机学院面向高年级本科生及研究生开设了《智能嵌入式技术》课程。课程旨在系统构建学生对智能嵌入式系统“从硬件到软件、从算法到应用”的完整知识体系，培养学生具备设计、开发与部署智能嵌入式产品(如智能机器人、边缘 AI 设备、工业视觉系统等)的综合工程能力。通过理论学习与项目实践相结合的方式，使学生能够在真实场景中运用所学知识，完成从系统选型、模型训练、软硬件协同优化到最终部署的全流程开发。

2. 研究现状

(1) 新工科建设背景下的教育改革驱动

“新工科”建设作为一项持续深化工程教育改革的重大行动计划，其核心在于推动学科交叉融合与创新能力培养，以回应新一轮科技革命与产业变革对工程技术人才提出的全新要求[1]。在这一宏观背景下，高等教育，特别是与信息技术密切相关的工程学科，面临着重构知识体系、更新教学内容与创新教学模式的紧迫任务。嵌入式系统作为信息技术与物理世界交互的关键载体，其课程教学的改革与升级，是践行新工科理念、培养适应未来智能产业发展需求人才的重要切入点。传统嵌入式系统教学在理论传授与实践能力培养上的脱节，以及在内容上对前沿技术(如人工智能)融合的不足，已成为制约其满足新工科要求的主要瓶颈[2]。因此，探索并实施面向 AI 融合的嵌入式系统课程改革，不仅是技术发展的自然结果，更是高等教育主动适应时代变革、提升人才培养质量的战略选择。

(2) 技术与产业融合发展的迫切需求

当前，我们正处在一个由人工智能、物联网和机器人技术共同驱动的智能时代。嵌入式系统的内涵与外延正发生深刻演变：它已从功能单一、资源受限的控制单元，演进为集成多模态感知、本地化智能推理、自主决策与实时控制能力的综合性智能终端[3]。从自动驾驶、工业机器人到智能家居，智能嵌入

式系统正以“感知–决策–控制”的闭环模式，深度赋能千行百业。这一产业变革的底层逻辑是人工智能技术与嵌入式系统的深度融合，对既精通传统嵌入式软硬件开发，又掌握AI模型部署优化与系统集成能力的“全栈式”复合型工程师产生了迫切需求[4]。这种强烈的市场需求，明确要求高校的嵌入式课程必须超越传统硬件与底层编程的范畴，系统性地融入人工智能技术，以培养学生的跨学科综合应用能力[5]。

(3) 现有教学体系与模式面临的挑战与局限

审视当前高校的课程体系，一个突出的问题是传统嵌入式教学与人工智能教学之间存在明显的“知识断层”。传统课程往往侧重于微控制器架构、接口编程与RTOS，教学实践或偏重理论，或局限于基础硬件设计，与前沿智能应用场景结合不紧密[5] [6]。而人工智能课程则多聚焦于算法理论与云端训练，极少涉及在资源受限的嵌入式环境中进行算法部署与优化的实践[2]。这种割裂导致学生能力结构失衡：嵌入式方向的学生难以驾驭复杂AI模型的落地，而AI方向的学生则缺乏对算法物理约束(功耗、实时性)及其与物理世界交互的深刻理解[7]。现有的教学模式在培养学生解决“软硬协同”复杂工程问题能力方面存在明显不足[3]。

(4) 已有教学改革探索及其深化空间

针对上述挑战，教育界已展开一系列改革探索，主要集中在以下几个方向：一是强化产教融合与项目实践，通过引入企业真实项目或竞赛题目，以“项目导向”和“课赛结合”来提升学生的工程实践能力和创新思维[4] [7]。二是革新教学方法与内容，例如采用基于CDIO工程教育理念的全生命周期教学模式[7]，或利用MOOC、虚拟仿真(如Proteus)等技术丰富教学手段与资源[8]。三是尝试跨学科内容整合，部分改革开始将嵌入式系统与机器人(ROS)、物联网等应用领域结合，或在实验中引入AI元素[9]。这些探索取得了积极成效，如提升了学生的硬件设计能力、工程素养和学习主动性[4] [5]。

然而，现有改革仍存在深化空间。多数尝试尚未系统性地将人工智能作为核心知识模块深度融入嵌入式课程体系。它们往往是在应用层调用AI服务，或仅将AI作为孤立章节介绍，未能贯穿“算法设计–模型优化–硬件部署–系统集成”的全链条能力培养。课程内容在反映异构计算架构(CPU/GPU/NPU)、边缘AI推理框架等前沿技术方面仍显不足。因此，构建一个深度融合AI与嵌入式技术、覆盖从底层硬件到顶层智能应用的全栈知识体系，并设计相应的系统性实践教学方案，是当前教学改革需要突破的关键[2] [3] [9]。

(5) 本研究课程的定位与创新导向

在此背景下，本研究进行的《智能嵌入式技术》课程改革，旨在直接回应新工科建设要求与产业融合趋势，针对现有教学体系的不足进行系统性革新。本课程定位并非对传统内容的简单修补，而是致力于构建一座连接“嵌入式硬件工程”与“人工智能算法”的桥梁[2]。其核心创新导向在于：第一，系统性知识融合：打破学科壁垒，以“智能嵌入式系统”为核心，全栈式整合异构计算、边缘AI部署、机器人系统等前沿内容[5] [7]。第二，前沿实践导向：强调“硬件感知”的算法优化与系统级设计思维，培养学生解决在资源约束下实现智能功能的复杂工程问题的能力[7]。第三，深度项目驱动：设计贯穿“感知–决策–控制”闭环的综合性项目(如智能机器人)，使学生经历从需求分析到系统集成的完整工程实践[7]。本研究即是对这一改革理念、课程体系构建与教学实施路径的系统性探索与实践总结。

3. 智能嵌入式技术知识体系

构建一个逻辑清晰、覆盖全面且紧跟技术前沿的知识体系，是《智能嵌入式技术》课程成功实施的基础。本课程知识体系并非嵌入式技术与人工智能技术的简单叠加，而是以“系统思维”为主线，以“实现一个智能嵌入式产品”为最终目标，进行深度融合与结构化重构。课程内容被整合为五个循序渐进的

层次：系统认知与概述、硬件平台与基础、软件栈与开发环境、智能算法与部署优化、典型应用与系统集成。该体系引导学生从宏观认识到微观实现，从理论奠基到工程实践，逐步构建完整的智能嵌入式技术知识图谱。

课程知识体系呈现为金字塔结构，底部为基础原理支撑，顶部为综合应用与创新，中间层为承上启下的核心技术。

第一层：系统认知与概述

本部分旨在建立学生对智能嵌入式系统的全局视野。首先阐释其定义、四大核心能力(实时感知、数据处理、自主决策、协同控制)及其相较于传统嵌入式系统与纯云端智能的显著优势(低延迟、高可靠、隐私安全、带宽高效)。通过剖析在自动驾驶、工业机器人、智能终端等领域的典型应用，使学生理解技术价值与产业意义。进而，系统解析智能嵌入式系统的五大组成部分：异构化硬件计算平台、层次化软件栈与框架、协同优化的智能算法、高效可靠的系统通信、系统化的开发流程。本部分强调各组件间的协同与依赖关系，奠定学生的系统级思维基础。

第二层：硬件平台与基础

硬件是智能功能实现的物理基石。本部分深入探讨为突破“内存墙”与“算力墙”而发展的异构计算架构。核心内容包括：作为控制与调度中枢的嵌入式 CPU(重点包括 ARM Cortex-A 系列的多核与 SIMD 技术、Cortex-M 系列的 Helium 向量扩展)；作为大规模并行计算引擎的 GPU；专为神经网络计算设计的 NPU/TPU；擅长高实时性信号处理的 DSP；提供高度定制化加速能力的 FPGA/SoC。此外，详细讲解支撑计算单元高效运行的存储架构(多级缓存、AI 专用紧耦合内存、DDR、eMMC/UFS)，以及连接传感器与执行器的关键外部接口(如 MIPI CSI/DSI、USB、PCIe、CAN 等)。最后，通过对比分析 NVIDIA Jetson 系列、瑞芯微 RK3568/RK3588、华为昇腾系列、ARM Cortex-M85/M55 等主流嵌入式 AI 芯片平台，培养学生根据性能、功耗、成本及生态进行硬件选型的工程能力。

第三层：软件栈与开发环境

软件是连接硬件与智能算法的桥梁，决定了系统开发效率与运行可靠性。本部分覆盖从底层到上层的完整软件生态。在操作系统层面，对比讲解资源受限场景下的实时操作系统(如 FreeRTOS 等)与功能丰富的嵌入式 Linux (包括 Bootloader 如 U-Boot、内核配置与编译、驱动开发、根文件系统构建)。针对现代异构多核芯片，重点介绍混合关键系统部署方案，包括非对称多处理(AMP)与对称多处理(SMP)模式，以及轻量级虚拟化/分区技术(如 Jailhouse、Bao Hypervisor)在实现安全隔离与实时保障中的应用。在开发框架层面，深入讲解机器人领域事实标准 ROS2 (基于 DDS 的分布式通信机制)，以及面向消费级设备的 Android 和面向万物互联的 OpenHarmony 系统。此外，课程引入现代软件工程实践，如使用 Docker 进行环境容器化以保障一致性，以及 CMake 等构建工具进行项目管理。

第四层：智能算法与部署优化

本部分是课程“智能”属性的核心体现。首先回顾深度学习基础，包括卷积神经网络、Transformer 及强化学习网络，但重点迅速转向嵌入式场景下的独特挑战与解决方案，即“模型轻量化与优化”。系统讲授后训练量化与量化感知训练，将模型参数从 FP32 降至 INT8/INT4，大幅减少存储与计算开销；讲解网络剪枝技术，移除冗余神经元或连接，得到稀疏高效的结构；介绍知识蒸馏方法，利用大模型(教师网络)指导小模型(学生网络)训练，在减小规模的同时保持性能。同时，涵盖 ONNX 作为中间表示格式在跨框架模型转换与图优化中的作用。在部署层面，详细剖析主流推理框架的工作流程，如 NVIDIA TensorRT 的层融合与精度校准、TensorFlow Lite 的算子转换与委托机制，以及华为 CANN、瑞芯微 RKNN 等硬件厂商专用工具链的使用。本部分强调“硬件感知”的算法设计，培养学生针对特定加速单元(如 NPU

指令集)进行模型调优的能力。

第五层：典型应用与系统集成

本层旨在将前四层知识融会贯通，通过典型应用案例完成系统级整合。课程选取计算机视觉和自然语言处理两大最具代表性的 AI 应用方向。在视觉方面，从 OpenCV 基础处理讲起，重点结合 YOLO 系列等目标检测模型，讲解从视频采集、图像预处理、模型推理到结果后处理的完整流水线，并涉及视频编解码与传输技术。在语音与语言方面，涵盖音频信号采集、前端处理(降噪、增强)，以及语音识别、语音合成和轻量化大语言模型(如 ChatGLM、Llama 等)在边缘设备的部署策略与案例。最终，所有技术汇聚于“具身智能机器人开发”这一综合性应用。学生将学习机器人本体的软硬件分层架构(底层 MCU 控制与上层 MPU 智能)，掌握激光与视觉 SLAM 进行地图构建与定位的原理与部署，并实践基于深度学习或深度强化学习的自主导航与动态避障算法。通过该完整案例，学生亲身体验从传感器数据采集、多模态融合、智能决策生成到物理执行器控制的“感知－决策－控制”闭环实现，形成对复杂智能嵌入式系统从设计到实现的完整认知。

该知识体系以系统化、层次化、应用导向为核心特征，既覆盖了智能嵌入式领域的基础理论与关键技术，又突出了跨硬件、软件、算法与通信的协同设计思维。通过理论讲解、案例分析与实践项目相结合的教学方式，旨在培养学生解决复杂工程问题的综合能力，为其在人工智能与嵌入式融合的快速演进中奠定坚实的学术基础与工程素养。

4. 教学实施方案

为确保知识体系的有效传递与学生工程能力的实质性提升，《智能嵌入式技术》课程实施了“三维驱动、学做赛一体”的教学方案。该方案将系统化理论授课、贯穿学期的项目式实践与以赛促学的创新拓展深度融合，形成环环相扣、相互强化的教学闭环，旨在培养学生从原理认知、技术实现到系统创新的全链条能力。

4.1. 教学授课：模块精讲、案例贯穿与专题研讨

理论授课是构建知识体系的基石。课程采用“模块化精讲 + 案例深度剖析 + 专题互动研讨”三位一体的教学模式，严格遵循五层知识体系展开。

模块化精讲：每个核心知识模块均采用“原理－技术－工具－实践”四步法讲解。例如，在 “硬件平台与基础”模块，不仅阐释 CPU/GPU/NPU/DSP/FPGA 的异构架构原理，更结合智能嵌入式系统实践中的 Nvidia Jetson Orin 与 RK3588 等开发平台，并现场演示使用 tegrastats、rknn-toolkit 等工具监控算力利用率，使抽象架构具象化。在“智能算法与部署优化”模块，则以一个具体的轻量化模型(如 MobileNetV2)为例，完整演示从 PyTorch 训练、ONNX 导出、到使用 TensorRT 进行 INT8 量化与层融合优化，最终在嵌入式平台实测延迟与功耗的全链路流程，让学生直观量化优化技术的收益。

案例深度剖析：为每一关键技术层配备源自产业与前沿的实战案例。在讲解软件栈(ROS2)时，深度剖析一个自动驾驶仿真案例，解读其感知、定位、规划节点如何通过 DDS 主题与服务进行低延迟、高可靠通信。在讲解模型轻量化时，则剖析一个视觉目标识别的项目，展示如何通过通道剪枝与量化感知训练，将模型压缩后部署至各种加速模块，实现实时的目标检测。这些案例直接来源于教材的实践章节与行业最新应用，确保了教学内容的时效性与工程相关性。

专题互动研讨：设置开放性、决策性专题，驱动高阶思维。例如，提出“为满足 10 W 功耗约束下的仓储机器人，如何选型 SoC (对比 Nvidia Jetson Nano 与 RK3588)并设计其 SLAM 与避障软件架构？”或“在边缘设备部署微调后的 Llama 2-7B 模型，有哪些可行的量化、裁剪与编译优化策略来保证响应速

度？”学生需基于课程所学，进行资料调研、技术选型论证与方案设计，并在课堂进行分组辩论与陈述。教师角色从讲授者转变为引导者与评判者，有效培养了学生的系统权衡能力与技术创新思维。

4.2. 课程作品实践：全周期、跨层级的项目式工程训练

课程作品实践是知识融合与能力转化的核心环节。学生以 3~4 人小组形式，在一个学期内完成一个覆盖“感知－决策－控制”闭环的智能嵌入式系统项目，其进程与知识讲授同步，严格遵循产品开发流程。

项目选题与设计(对应知识层：系统认知&硬件/软件选型)：项目方向限定于典型应用领域(如自主移动机器人、智能视觉交互终端、边缘 AI 网关)，但具体功能需学生自主创新定义。首月需完成《项目立项报告》，内容必须包含详细的需求分析、基于性能/功耗/成本的硬件平台对比选型(如对比使用 Cortex-M55 微控制器还是 Cortex-A55 SoC)、软件框架论证(如选用 ROS2 还是 OpenHarmony)，并提交详细的物料清单与开发计划。此阶段强制学生运用第一、二层知识进行系统性设计思考。

模块开发与集成(对应知识层：软件栈、算法部署)：项目中期，学生并行开展各模块开发。例如，在一个“基于多模态感知的跟随机器人”项目中，学生需：在嵌入式 Linux 上开发摄像头与激光雷达驱动；利用 OpenCV 进行传感器数据预处理；训练并部署一个轻量化的模型(运用量化、剪枝技术)进行人体检测与跟踪；在 ROS2 中集成导航栈，实现基于动态窗口法的局部路径规划；最后，将所有模块通过 ROS 节点进行集成，实现稳定跟随。此阶段深度锤炼第三、四层知识的编码与调试能力。

系统联调与优化(对应知识层：系统集成与优化)：后期工作聚焦于系统整体性能提升与稳定性打磨。学生需进行严格的测试：测量端到端延迟、优化内存占用、提升算法精度与鲁棒性，并可能进行硬件的二次调整(如增加散热、优化供电)。最终需提交完整的项目实物、源代码、技术报告及演示视频，并进行期末答辩。通过此全过程，学生将五层知识体系融会贯通，完整经历了产品从 0 到 1 的工程化实现，培养了至关重要的系统集成与问题解决能力。

4.3. 参加相关比赛实践：以赛促学、以赛促创的创新能力淬炼

学科竞赛是检验学习成果、激发创新潜能、对接产业前沿的最高效平台。课程积极构建“课程作品－竞赛项目－创新成果”的转化通道，组织并指导学生参与一系列高水准专业赛事。

竞赛对接与选拔：课程重点瞄准与智能嵌入式技术紧密相关的权威竞赛：包括中国国际大学生创新大赛、“挑战杯”全国大学生课外学术科技作品竞赛、计算机系统能力大赛、嵌入式芯片与系统设计竞赛等；由核心企业主导的华为昇腾 AI 创新大赛、NVIDIA Jetson 开发者挑战赛等；以及聚焦机器人技术的机器人竞赛等。这些赛事的赛题往往直接针对边缘计算、机器人、物联网等前沿应用，与课程内容高度契合。

备赛支持与成果转化：教学团队为参赛队伍提供全方位赋能：包括赛题技术路线解读、开放实验室与硬件平台支持、过往获奖作品的经验复盘会，以及模拟路演与答辩训练。许多优秀的课程作品在经过竞赛的深度打磨后，性能与创新性得到极大提升，进而转化为发明专利、学术论文或创业项目的雏形。例如，一项课程中的“巡检机器人”作品，经过优化后参加了“挑战杯”竞赛，其创新的轻量化视觉识别模型与低功耗导航方案获得了评委的高度认可。这种“以赛促学、以赛促创”的模式，不仅让学生在极限压力下巩固了技术，更锻炼了其团队协作、项目管理和创新策划等综合素质，显著提升了课程的高阶性与挑战度。

5. 结束语

本课程建设是对人工智能与嵌入式系统融合趋势的积极回应，旨在培养精通“算法－硬件－软件”

协同设计与系统集成的复合型人才。课程构建了覆盖系统认知、硬件基础、软件栈、算法部署、应用集成五层的全栈知识体系，弥合了传统专业分野。

教学实施采用“理论－实践－竞赛”三维驱动模式：通过模块化精讲与案例剖析夯实理论基础；以贯穿学期的综合性项目驱动学生完成从设计到实现的完整工程训练；鼓励参与高水平学科竞赛，淬炼创新与解决极限问题的能力。该模式成效显著，学生作品在多项权威赛事中屡获佳绩，毕业生在智能硬件、机器人等领域展现出强大竞争力。

面对技术快速演进，课程将持续迭代：及时纳入类脑计算、存算一体等前沿内容；深化与行业领军企业的产教融合，共建实验室并引入产业级开发流程；延伸培养链条，推动优秀成果向毕业设计、科研及创业项目转化。

本课程建设的目的不仅为学生提供了清晰的学习路径，也为深度交叉融合的课程建设提供了可推广的范式，致力于为国家关键科技领域培养优秀后备力量。

基金项目

1) 结合人工智能内容的嵌入式系统课程教学改革与实践，2025 年度广东省高等教育教学研究和改革项目(2025-474)；

2) 智能嵌入式技术，2025 年华南理工大学专创融合型深度学习课堂项目(x2jsC9260390)。

参考文献

[1] 管芳, 刘涛, 赵中华, 等. 新工科背景下的嵌入式系统课程教学改革研究[J]. 大学教育, 2025(6): 55-60.

[2] 毕盛, 董敏, 冼进, 等. 结合人工智能技术的嵌入式系统教学改革[J]. 当代教育实践与教学研究(电子刊), 2024(10): 73-76.

[3] 梁晶, 吴银琴, 陈锟. 面向解决复杂工程问题能力培养的嵌入式系统教学改革探索[J]. 计算机教育, 2024(9): 146-149.

[4] 漆强, 周建华, 刘子骥, 等. 基于新工科理念的“嵌入式系统设计”课程改革和创新[J]. 实验室研究与探索, 2025, 44(6): 162-166.

[5] 姜开永, 胡赟, 周小明. “嵌入式系统应用开发”应用型课程实践教学模式研究[J]. 物联网技术, 2025, 15(7): 159-162.

[6] 刘伟伟, 张艳玲, 李琳琳, 等. 嵌入式系统及应用课程的实践教学改革[J]. 计算机教育, 2024(8): 109-113.

[7] 游凤芹, 曾刚, 许超. 基于 CDIO 工程教育理念和产教融合导向的嵌入式系统课程教学探索[J]. 计算机教育, 2025(6): 178-182.

[8] 侯志伟, 杜青青, 包理群. 融入虚拟仿真的 STM32 应用开发实验教学改革与实践[J]. 物联网技术, 2024, 14(7): 147-151.

[9] 李磊, 高学, 秦慧平, 等. 嵌入式人工智能系统实验课程改革与探索[J]. 实验室研究与探索, 2025, 44(3): 147-152.

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(6), 355-362
Published Online December 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.26035

Hans汉斯

虚实结合的机载软件自动化测试技术研究与应用

张　絮，韩　启，王　媛

中航工业一飞院，陕西　西安

收稿日期：2026年2月2日；录用日期：2026年5月1日；发布日期：2026年5月27日

摘　要

为解决机载软件测试中存在的测试覆盖不充分、测试效率低等问题，文章从软件功能特点和测试需求出发，利用数字化驱动的自动测试手段，进行虚实结合的机载软件自动化测试技术研究，提出基于半实物仿真测试环境的自动化测试系统，在保证目标软件运行真实性同时，实现自动化测试全过程，并以实际机载软件测试应用进行工程实践，结果表明通过该测试系统可有效提高机载软件测试效率和质量，确保机载软件可靠、按期交付。

关键词

虚实结合，自动化测试，半实物仿真，测试效率

Research and Application of Hybrid Virtual-Physical Automated Testing Technology for Avionics Software

Xu Zhang, Qi Han, Yuan Wang

AVIC the First Aircraft Institute, Xi'an Shaanxi

Received: February 2, 2026; accepted: May 1, 2026; published: May 27, 2026

Abstract

To address the issues of insufficient test coverage and low testing efficiency in avionics software testing, this paper conducts research on virtual-physical integrated automated testing technology

文章引用: 张絮, 韩启, 王媛. 虚实结合的机载软件自动化测试技术研究与应用[J]. 嵌入式技术与智能系统, 2025, 2(6): 355-362. DOI: 10.12677/etis.2025.26035

for avionics software by leveraging digitization-driven automated testing methods, starting from software functional characteristics and testing requirements. An automated testing system based on a semi-physical simulation testing environment is proposed, ensuring the authenticity of target software execution while achieving full-process automation. The system is validated through practical engineering applications in avionics software testing, with results demonstrating that it significantly improves testing efficiency and quality, thereby ensuring reliable and on-time delivery of avionics software.

Keywords

Hybrid Virtual-Physical, Automated Testing, Semi-Physical Simulation, Test Efficiency

1. 引言

随着机载系统综合化程度不断提高，机载软件规模的逐年增长，软件测试对软件质量的重要作用愈加凸显。机载软件配置项、系统级测试主要在系统试验室目标机环境中进行，由于外部环境的局限性，往往在较多异常及边界情况下测试的覆盖不够充分[1]，可能造成较多软件故障隐患，同时软件测试人员手工执行测试效率低，制约了测试工作的推进，给机载软件研制及交付带来了较大的压力，因此传统的软件测试手段已不能满足较短周期内高复杂度和高规模的机载软件测试要求了。

2. 机载软件特点

航空机载系统普遍存在交联系统设备数量庞大，测试场景复杂，同时涉及较多人机交互行为。以综合显示控制应用软件为例，作为航空电子系统的显示终端，为飞行机组提供综合显示信息，完成综合控制。其显示画面类型多且变化多样、交联逻辑复杂、操作形式多变，对其可靠性要求越来越高[2]，在软件测试中，应对其任务画面的显示和控制逻辑功能、接口等进行全面测试。

3. 机载软件自动化测试环境

航空机载软件大多数属于重要级以上软件，必须在目标计算机运行环境中进行相应的黑盒测试，对此，机载软件自动化测试研究需建立在虚实结合的半实物仿真测试环境基础上。

半实物仿真测试环境采用真实机载系统设备，外围环境为系统仿真的数字化模型，以及总线监控、飞行战场环境、自动测试引擎等软件测试环境，通过机载网络板卡与机载系统设备交联，具有轻量化、部署灵活、可扩展能力强、能够快速构建等特点，由于半实物仿真测试环境采用了真实机载系统设备，保证了目标软件运行环境的真实性，将自动化测试技术与半实物仿真测试环境相结合，包括对机载软件显示画面和控制逻辑等的自动测试，从而可实现虚实结合的机载软件自动化测试需求。

4. 虚实结合的机载软件自动化测试系统设计

虚实结合的机载软件自动化测试系统能够帮助测试人员快速与半实物仿真测试环境集成，灵活接入被测目标系统，支持根据软件需求快速或自动生成测试用例，自动生成可执行脚本并执行，自动记录测试结果，自动生成测试报告及数据管理等功能[3]，进行便捷、高效的虚实结合的软件自动化测试。

4.1. 自动化测试系统架构

自动测试系统架构采用分层式设计架构，分为测试用例设计层和测试用例执行层，在测试用例设计层，采用关键字驱动技术，通过对需求进行分析，基于需求快速设计基于关键字的测试用例集，可有效提高测试用例的复用性和可维护性[4]。在测试用例执行层，将测试用例依据映射表转译为测试脚本，通过测试引擎执行测试脚本，生成测试报告。

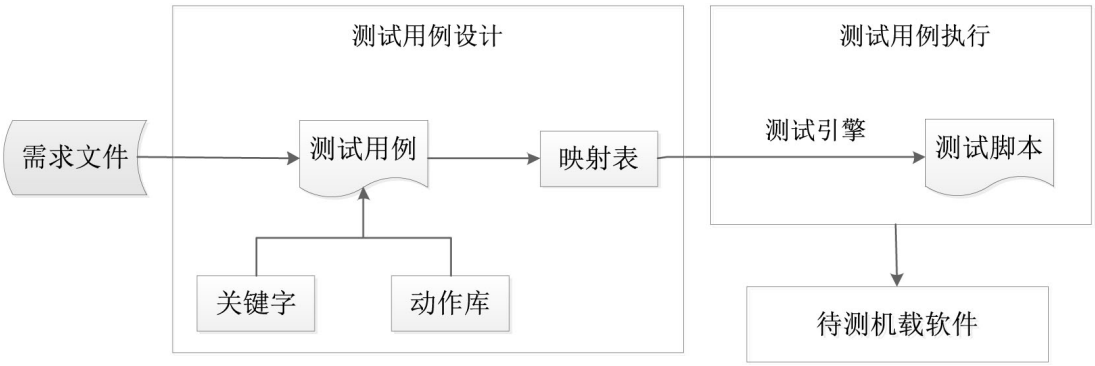


Figure 1. Automated testing system architecture design diagram
图 1. 自动测试系统架构设计图

自动测试系统架构设计如图 1 所示，自动化测试首先从关键字开始定义，定义具有工程意义的对象，如软件应用功能的信号定义“燃油量”、“飞行速度”等，测试用例设计同时定义一系列针对配置项、系统测试的动作库，如“设置”、“验证”等，利用关键字 + 动作即可进行测试步骤的编写，多个测试步骤形成测试逻辑，不同的测试逻辑对应不同的测试用例。

下一步，建立关键字与软件接口定义对应的映射表，确定二者的唯一对应关系，如设置速度 SET_SPEED 对应于 DEVxx.BLKxx.SIGxx.speed。

在测试用例中，通过关键字引用映射表中的接口定义信号，将其转换为可执行的测试脚本，当软件接口定义发生变化而功能不变时，只需修改相应的映射关系列表，不需要对测试用例进行更改，从而减少了测试用例的维护工作量[5]。

执行端中具有测试引擎，该引擎根据关键字从映射表中获取对应的接口定义信号信息，对每个步骤进行解析，每个步骤中都包含动作，最终对动作进行执行，同时，将每个动作中的接口定义信号再次打包为总线消息块，该消息块对应于接口定义信号组包后的消息块，用于总线传输。

最终，每个消息块都按照动作执行，包括发送、等待等，作用到待测软件中，同时对其响应进行检测判别，获得测试最终结果。

4.2. 自动化测试系统功能

建立如图 2 所示的机载软件自动化测试系统，在测试时，首先通过测试用例设计模块生成与需求对应的测试用例，并存储至数据库模块进行管理，在自动测试执行模块将测试用例转换为可执行测试脚本，通过接口模块与半实物仿真测试环境进行测试数据交互，获取测试脚本执行中数据激励或采集的接口定义信号及数值。对于具有显示画面的测试用例，可与外部图像识别模块通信，提供进行识别、判别所需的信息，并获取图像识别模块的返回信息。最后通过测试报告模块生成符合要求的测试报告。在测试过程中，可通过数据监控模块对测试过程数据进行可视化监控。

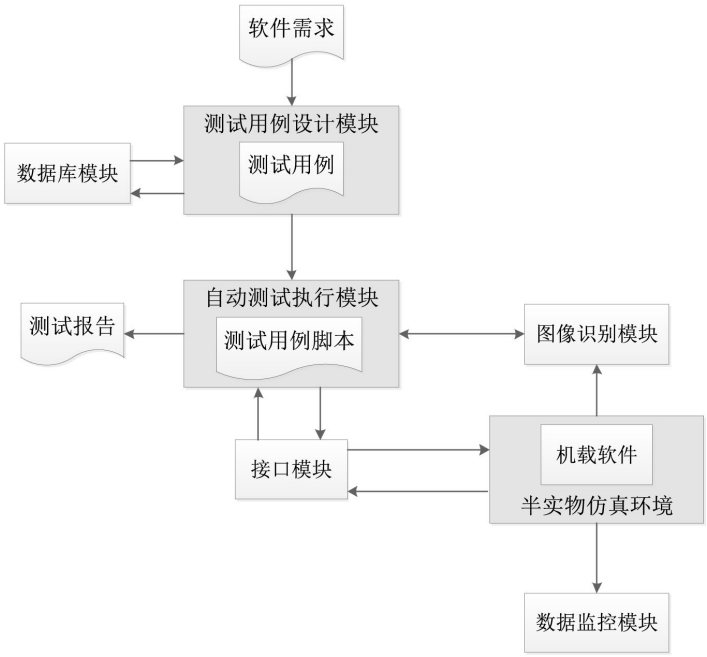


Figure 2. Avionics software automated testing system functional architecture
图 2. 机载软件自动化测试系统功能结构图

5. 机载软件自动化测试平台构建

机载软件自动化测试平台可支持基于关键字的快速测试用例设计，并自动生成可执行测试脚本，能够提供被测软件真实目标环境及外部接口及工作环境的模拟功能，支持机载软件显示画面、控制逻辑等功能、接口测试及性能测试，具有测试管理、测试驱动以及测试报告生成等主要功能，能够对被测数据实时监控，并且根据不同测试环境需求进行快速集成、适配[6]。

测试平台由软件部分和硬件部分组成，其功能主要由测试用例设计工具、自动测试执行管理工具、接口适配工具组成，如图 3 所示。

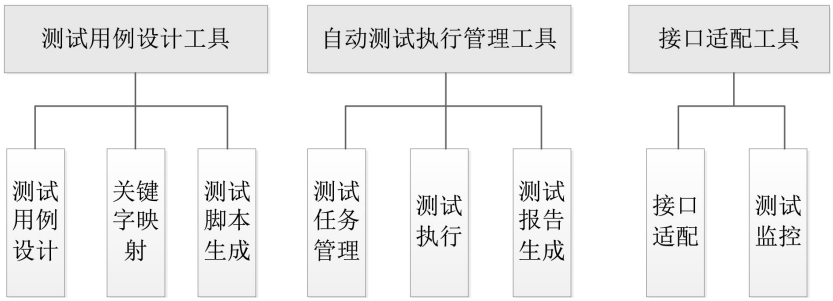


Figure 3. Avionics software automated testing platform functional composition
图 3. 机载软件自动化测试平台功能组成

1) 测试用例设计工具

测试人员可利用测试用例设计工具进行用例设计，包括多种标准动作库、处理流程等，根据测试需求定义关键字，扩展动作，完成用例设计。支持关键字映射关系生成，并将测试用例解析为可自动执行的测试脚本。

2) 自动测试执行管理工具

支持测试计划制定，导入待执行的测试用例脚本，依据测试任务管理测试用例执行[2]，测试完成后自动生成测试报告。

3) 接口适配工具

接口适配工具可实现对全数字虚拟测试环境和半实物仿真测试环境的适配以及测试监控功能。

全数字仿真环境指机载软件部署运行于本地 Windows 系统的电脑环境中，一般在基于真实目标机的半实物仿真测试环境测试之前，在全数字虚拟环境中对软件程序进行调试和预测试，可尽早发现软件错误，由于自动化测试系统与被测环境的解耦设计，同一套测试用例同时可同时应用于两种测试环境中，进一步提高了软件测试效率和质量。

当软件测试运行在虚拟测试环境中时，利用该模块进行接口适配，完成对机载软件的数据激励和采集，从而实现自动测试回路；当需要在基于真实目标机测试环境中测试执行时，可通过一键式配置切换，实现对真实测试环境的数据激励，并通过数据采集或图像识别接口实现自动测试回路。在软件测试执行过程中实时对测试数据进行监控和回放。

测试平台运行工作原理如图 4 所示：首先在测试设计层实现对需求的分析，基于需求进行用例设计，依次生成测试用例、可执行测试脚本，在测试执行层建立测试任务，执行测试脚本，通过接口适配工具实现测试环境配置、数据激励和数据监控等功能，完成与虚拟或真实测试环境的通信，生成测试报告，最终完成整个自动测试过程。

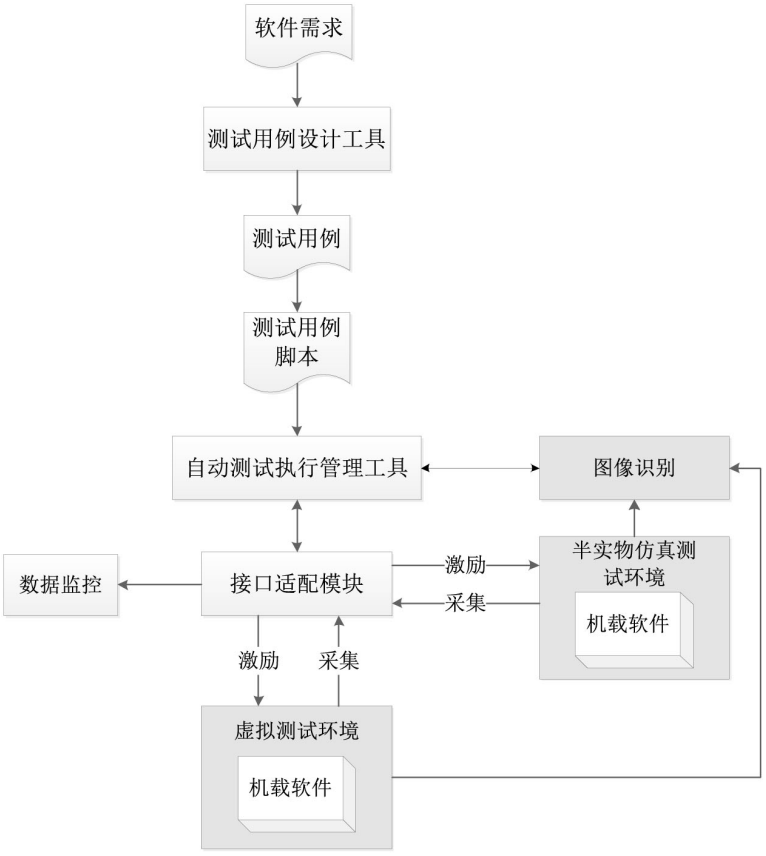


Figure 4. Working principle diagram of the test platform
图 4. 测试平台运行工作原理图

6. 机载软件自动化测试应用

在所构建的虚实结合的机载软件自动化测试平台上，选取航空电子系统某显示控制软件中的典型功能进行测试应用，包括测试用例设计、测试执行、测试报告生成等功能。

6.1. 测试用例设计

选取某系统子页面的转弯系统工作状态显示功能，作为软件中典型显示控制功能进行应用，该功能描述为：软件读取转弯系统工作状态信号，在转弯角度左下方正确显示工作状态字符，字符颜色根据信号中的“系统显示颜色”确定。

在基于关键字的测试用例编辑界面编写该功能对应的测试用例如图 5 所示，包含测试用例名称、用例说明、用例步骤及预期结果等，其中验证步骤采用图像识别对转弯状态的字符和颜色进行自动识别测试。

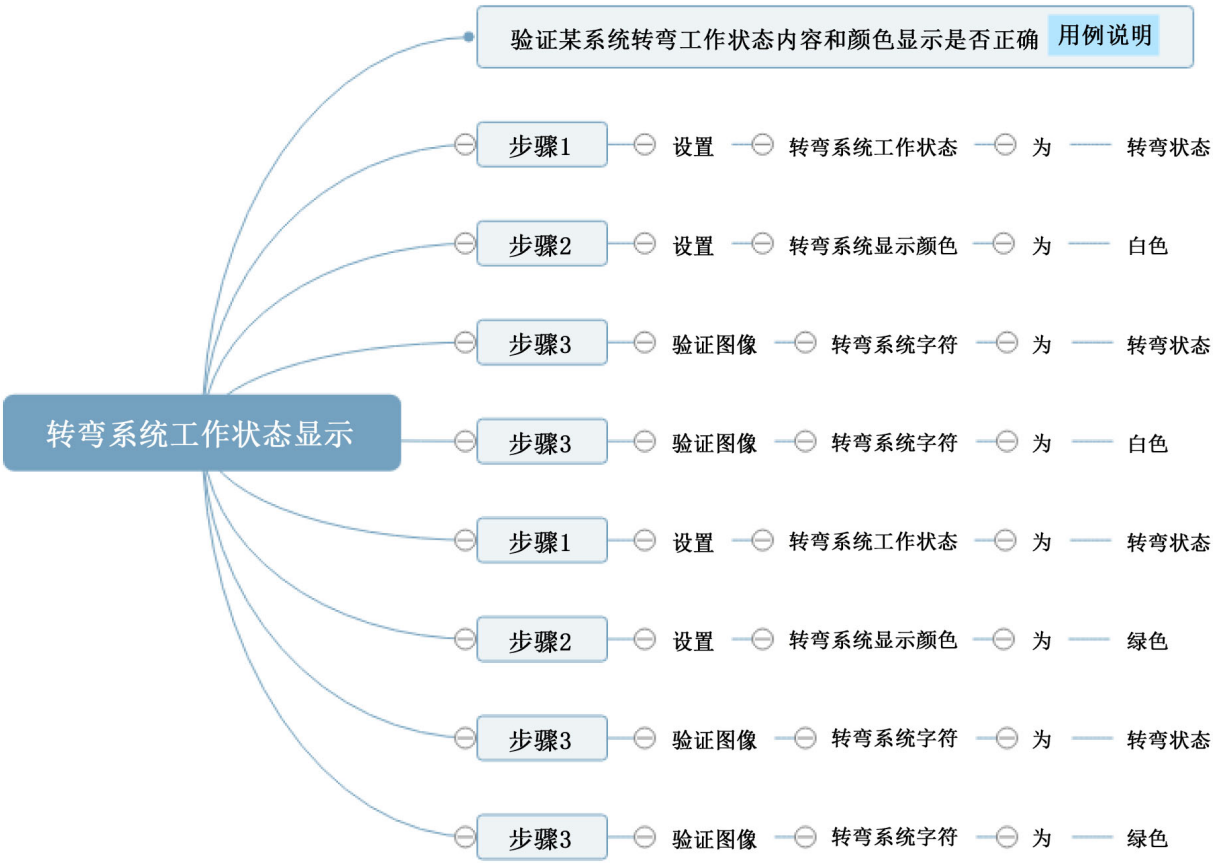


Figure 5. Steering system operating status display test case
图 5. 转弯系统工作状态显示测试用例

6.2. 测试用例执行

启动自动测试管理执行工具，建立相应的工程和测试任务，点击“运行”按键执行基于图像识别的自动测试，右侧显示测试过程中显示每一测试步骤运行情况，包括验证对象、期望值、识别结果图片和验证结果等，如图 6 所示，该测试用例所有步骤均通过，证明该软件功能满足相应设计要求。

验证结果	通过
其他	
识别结果	转弯状态
结果图片	http://10.2.149.132:9000/image/1692924202612.jpg
用例号	IDCS-PZX_09-TEST001
需求号	
验证对象	起落架_转弯状态.value
期望值	转弯状态
[1]	
验证结果	通过
其他	
识别结果	白色
结果图片	http://10.2.149.132:9000/image/1692924203014.jpg
用例号	IDCS-PZX_09-TEST001
需求号	
验证对象	起落架_转弯状态.color
期望值	白色

Figure 6. Comprehensive test result information
图 6. 测试结果详细信息

图 7 为测试结果实际截图(红色部分为被测元素“转弯状态”所打标记)，通过在测试结果详细信息处点击结果图片进行人工核查与问题确认。

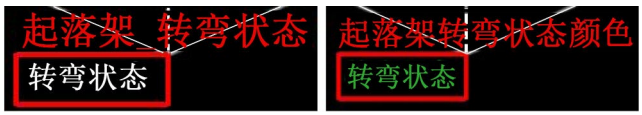


Figure 7. Test results screenshot
图 7. 测试结果截图

6.3. 测试总结

测试结束后，在相应路径下查看自动生成的测试记录及测试报告内容，生成报告格式满足相应标准要求，节省了测试人员编写报告的时间，缩短了测试周期。

7. 应用结果对比

针对以上测试过程，对传统手工测试和自动化测试在测试设计、测试执行和测试报告编写时间的数据进行比较，具体见图 8 所示。

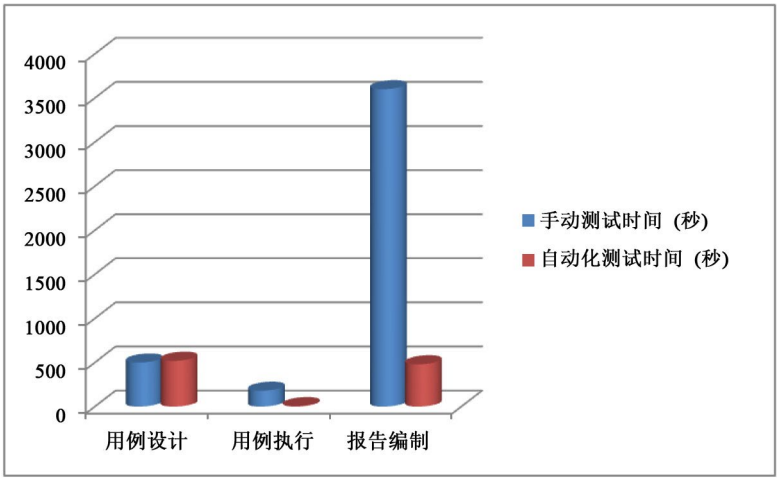


Figure 8. Test time comparison chart for different test cases
图 8. 测试时间对比图

通过实际工程应用表明，手工测试和自动化测试均能正确得到测试结果，根据图 8 数据对比，采用自动化测试比传统手工测试的总时间减少了约 76%，很大程度提高了软件测试效率，减少了测试人员的工作量，具有明显优势。

8. 结束语

虚实结合的机载软件自动化测试技术利用数字化驱动的自动测试手段，改变了传统的测试模式，所建立的测试系统在保证了测试结果有效性的同时解决了需求覆盖不全面的问题，利用自动化测试手段有效缩短了测试周期、提高软件测试效率，高效、便捷地辅助测试人员开展测试工作，确保了机载软件可靠、按期交付。

参考文献

[1] 郭旺, 丁晓明, 唐海鹏, 等. 半实物环境下嵌入式软件通用测试平台研究[J]. 西南大学学报(自然科学版), 2015(4): 62-66.

[2] 段海军, 赵根学, 陈福, 等. 航空电子设备自动测试系统的软件架构设计[J]. 计算机测量与控制, 2016, 24(9): 167-169.

[3] 夏佳佳, 邹毅军, 周江伟, 等. 嵌入式软件自动化测试系统研究[J]. 计算机测量与控制, 2016, 24(4): 22-25.

[4] 王军, 孟凡鹏. 基于关键字驱动的自动化测试研究与实现[J]. 计算机工程与设计, 2012, 33(9): 3652-3656.

[5] Alsmadi, I. (2008) Building a GUI Test Automation Framework Using the Data Model. VDM Verlag.

[6] 李碧涵, 胡益诚. 机载嵌入式软件的自动化测试架构设计[J]. 电脑编程技巧与维护, 2019(6): 13-15.

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(6), 363-374
Published Online December 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.26036

Hans汉斯

面向电子木琴自动演奏的MIDI信息智能转换算法研究

周涵艺[1,2]，刘卫玲[2,3*]，常晓明[2]

[1]太原理工大学计算机科学与技术学院，山西 晋中
[2]太原理工大学晓明研究室，山西 太原
[3]太原理工大学人工智能学院，山西 晋中

收稿日期：2026年4月7日；录用日期：2026年5月1日；发布日期：2026年5月27日

摘　要

针对电子木琴自动演奏系统中MIDI信息转换的关键问题，文章提出了两种核心算法：智能音高偏移优化算法和智能转换策略选择算法。智能音高偏移优化算法通过构建六维度加权评分体系，自动计算最优音高偏移量，将任意MIDI信息所包含的音域适配至木琴有效音域；智能转换策略选择算法基于轨道数、总音符数和主轨占比三个特征，通过规则评分自动选择全曲转换、主旋律轨转换或截取部分转换策略，以适应不同音乐类型和系统资源限制。在Python环境下开发了MIDI转换程序，实现了从MIDI信息到C语言头文件的完整转换流程。实验结果表明，两种算法能有效解决音域不匹配和策略选择问题，转换后的音高分布与原始旋律轮廓高度一致，验证了算法的有效性。研究为传统打击乐器的智能化改造提供了技术支撑。

关键词

电子木琴，MIDI转换，智能音高偏移优化算法，智能转换策略选择算法，自动演奏

Research on Intelligent Conversion Algorithm of MIDI Information for Automatic Performance of Electronic Xylophone

Hanyi Zhou[1,2], Weiling Liu[2,3*], Xiaoming Chang[2]

[1]College of Computer Science and Technology, Taiyuan University of Technology, Jinzhong Shanxi
[2]Xiaoming Research Laboratory, Taiyuan University of Technology, Taiyuan Shanxi

[*]通讯作者。

文章引用: 周涵艺, 刘卫玲, 常晓明. 面向电子木琴自动演奏的 MIDI 信息智能转换算法研究[J]. 嵌入式技术与智能系统, 2025, 2(6): 363-374. DOI: 10.12677/etis.2025.26036

[3]College of Artificial Intelligence, Taiyuan University of Technology, Jinzhong Shanxi

Received: April 7, 2026; accepted: May 1, 2026; published: May 27, 2026

Abstract

Regarding the key issue of MIDI information conversion in the automatic electronic xylophone playing system, this paper proposes two core algorithms: the intelligent pitch offset optimization algorithm and the intelligent conversion strategy selection algorithm. The intelligent pitch offset optimization algorithm constructs a six-dimensional weighted scoring system to automatically calculate the optimal pitch offset amount, adapting the pitch range contained in any MIDI information to the effective pitch range of the xylophone; the intelligent conversion strategy selection algorithm, based on three features - the number of tracks, the total number of notes, and the proportion of the main track - automatically selects the full piece conversion, the main melody track conversion, or the partial extraction conversion strategy to adapt to different music types and system resource limitations. The MIDI conversion program was developed in the Python environment, achieving a complete conversion process from MIDI information to C language header files. Experimental results show that the two algorithms can effectively solve the problems of pitch range mismatch and strategy selection, and the pitch distribution after conversion is highly consistent with the original melody contour, verifying the effectiveness of the algorithms. This research provides technical support for the intelligent transformation of traditional percussion instruments.

Keywords

Electronic Xylophone, MIDI Conversion, Intelligent Pitch Offset Optimization Algorithm, Intelligent Conversion Strategy Selection Algorithm, Automatic Performance

1. 引言

传统键盘式电子琴系统需通过用户物理敲击键盘完成音乐演奏，对用户的乐理知识和演奏技巧有一定要求。随着数字音乐技术的发展，如何实现乐器的自动化演奏、降低演奏门槛成为研究热点之一。MIDI作为一种数字音乐接口标准，能够将乐谱信息转化为数字指令，为乐器自动化提供了技术基础。

在电子木琴自动演奏系统中，核心问题之一是如何将 MIDI 信息转换为硬件可执行的指令。MIDI 信息记录了音符的音高、起始时间和持续时间，是“计算机能理解的乐谱”，但无法直接被微控制器(MCU)识别。由于 MCU 通常使用 C 语言进行编程，因此需要将 MIDI 信息转换为 C 语言头文件格式。

目前，MIDI 信息转换研究主要集中于音高识别[1]和主旋律提取[2]，面向有限音域乐器的自适应转换算法较少。现有方法多采用固定升降调或基于调性检测的整体平移，无法解决超出木琴音域(仅 2 个八度)的音符映射问题；同时，针对嵌入式系统资源限制的转换策略选择也缺乏系统研究。相比 MP3 等音频格式，MIDI 信息具有结构化的音符数据，便于算法处理，更适合实现智能化转换。

本文针对上述问题，提出两种核心算法：智能音高偏移优化算法和智能转换策略选择算法，前者通过六维度加权评分模型自动计算最优音高偏移量，将 MIDI 信息中的音域适配至木琴有效音域；后者基

于轨道数、总音符数和主轨占比，自动选择全曲转换、主旋律轨转换或截取部分转换策略。在 Python 环境下开发的转换程序，实现了从 MIDI 信息到 C 语言头文件的完整转换流程。本文中所提到的“MIDI 信息”均指标准 MIDI 文件所包含的结构化音乐数据。

本文后续章节安排如下：第二章介绍两种算法的理论模型；第三章阐述 MIDI 转换程序的实现；第四章对算法进行验证与评估；第五章总结全文。

2. MIDI 信息智能转换算法的数学模型

从 MIDI 信息到 C 头文件转换的总体思路，如图 1 所示。

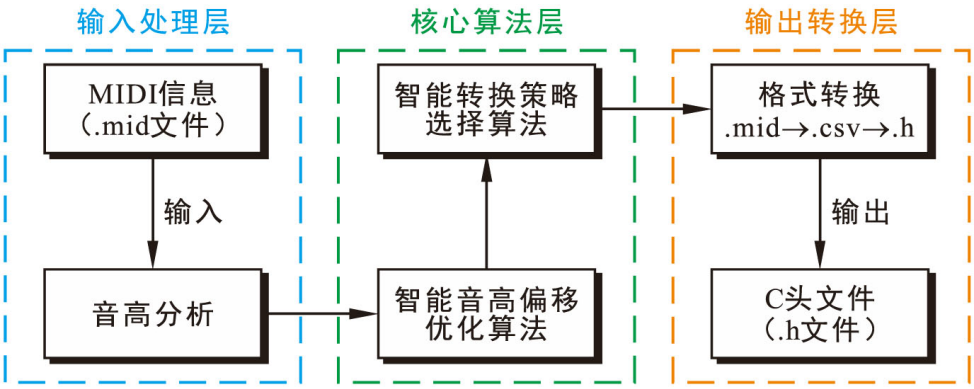


Figure 1. Overall conceptual diagram for MIDI information conversion
图 1. MIDI 信息转换总体思路图

图 1 中，输入的 MIDI 信息首先经过音高分析，统计各音符的音高及出现频率；然后由智能音高偏移优化算法将音域适配至木琴有效范围；再经智能转换策略选择算法根据音乐特征决定最优转换策略；最后通过格式转换生成控制器可执行的 C 语言头文件[3]。

2.1. 智能音高偏移优化算法的数学模型

智能音高偏移优化算法是一种多目标加权评分优化算法，目标是自动找到最优音高偏移量 d^*，使 MIDI 信息中的音域最大限度适配电子木琴音域(MIDI 编号 43~67) [4]。

该算法的核心目标为最大化综合评分函数，找到最优音高偏移量 d^*，计算公式如下：

$$d^* = \arg\max_{d\in[-48,48]} S(d) \tag{1}$$

式中，d^*：最优音高偏移量(半音数)，

d：候选音高偏移量(半音数)，搜索空间为[−48, 48]的整数，

$S(d)$：音高偏移量 d 的综合评分函数。

综合评分函数 $S(d)$采用六个维度的加权评分体系，每个维度从不同角度评估偏移量的优劣，计算公式为：

$$S(d)=\left(w_1\cdot R(d)+w_2\cdot E(d)+w_3\cdot C(d)+w_4\cdot G(d)+w_5\cdot M(d)+w_6\cdot P(d)\right)\times B(d) \tag{2}$$

式中，w_i：各维度权重比例，

$R(d)$：音域内比例，权重 w_1 为 0.25，

$E(d)$：音高分布熵，权重 w_2 为 0.20，

$C(d)$：中心音高贴近度，权重 w_3 为 0.15，

$G(d)$：音域利用率，权重 w_4 为 0.15，

$M(d)$：映射惩罚因子，权重 w_5 为 0.10，

$P(d)$：位置适应度，权重 w_6 为 0.15，

$B(d)$：八度移位奖励系数。

其中，$B(d)$为候选音高偏移量 d 的八度移位奖励系数，当且仅当 d 为 12 的整数倍(即整八度移位)时，予以奖励，计算公式如下：

$$B(d)=\begin{cases}1.05 & \text{if } d \bmod 12=0\\ 1 & \text{otherwise}\end{cases} \tag{3}$$

式(2)中，六个维度函数通过对应公式归一化到[0, 1]区间，保证不同量纲的评分可直接加权求和。以音域内比例函数 $R(d)$为例，说明公式形式，其归一化计算公式如下：

$$R(d)=\frac{\sum_{p\in P}\text{freq}(p)\cdot \text{II}(p+d\in V)}{\sum_{p\in P}\text{freq}(p)} \tag{4}$$

式中，II(·)：指示函数，条件成立时值为 1，否则为 0，

P：原始 MIDI 信息中所有不同音高的集合，

p：集合 P 中的某个音高值(MIDI 编号)，

freq(p)：原始音高 p 在 MIDI 信息中出现的次数(即频率)，

d：候选音高偏移量(半音数)，搜索空间为[−48, 48]的整数，

V：木琴有效音域区间，范围为[43, 67]，即 24 个半音(木琴音域理论最大跨度)。

六个维度函数用于评估候选音高偏移量 d 的适应度，每个函数的值域均归一化到[0, 1]区间，值越接近 1 表示该维度表现越优。六个维度函数意义，如表 1 所示。

Table 1. Function significance of six dimensions
表 1. 六个维度函数意义

NO.	六个维度函数	函数意义
01	音域内比例函数 $R(d)$	值为 0 表示没有音符直接落在有效音域内； 值为 1 表示所有音符均直接有效
02	音高分布熵函数 $E(d)$	值为 0 表示所有音符集中在单一音高； 值越接近 1 表示 15 个有效音高越均匀分布
03	中心音高贴近度函数 $C(d)$	值为 0 表示所有音符均在音域两端(最差)； 值为 1 表示所有音符均为中心音高 55
04	音域利用率函数 $G(d)$	值为 0 表示只使用一个音高(最集中)； 值为 1 表示使用全部 24 个半音(最广泛)
05	映射惩罚因子函数 $M(d)$	值为 0 表示所有音符都需要映射(最差)； 值为 1 表示没有音符需要映射(最佳)
06	位置适应度函数 $P(d)$	值为 0 表示整体平均音高严重偏离音域； 值为 1 表示整体平均音高在音域[43, 67]内

2.2. 智能转换策略选择算法的数学模型

智能转换策略选择算法是一种基于规则的三元决策算法，目标是通过分析 MIDI 信息的三个关键特征(轨道数、主轨占比、总音符数)，按照预设的评分规则，为三种转换策略计算综合得分，最终选择得分最高的策略，以在转换质量与资源占用之间找到最佳平衡[5]。

MIDI 信息具有多样性：部分为单轨纯旋律，部分为多轨复合音乐(含伴奏、打击乐)；音符数量也从几十到数千不等。而嵌入式系统存在内存限制，通常要求输入音符数小于 1,000 个。因此，需要据 MIDI 信息的特征自动选择最优转换策略。定义 MIDI 信息 M 的三个关键特征如下：

(1) 有效轨道数量 T：包含有效音符(velocity > 0 的 note_on 消息)事件的轨道数量；

(2) 总音符数 N：所有有效轨道中有效音符事件的总数；

(3) 主轨占比 R：有效音符数量最多的轨道占总音符数的比例；

对上述三个输入特征，算法预设三种转换策略处理输入的 MIDI 信息，如表 2 所示。

Table 2. Three conversion strategies
表 2. 三种转换策略

NO.	策略类型	转换逻辑	适用场景
01	全曲转换 S_1	保留 MIDI 信息所有有效轨道的有效音符	单轨简单音乐
02	主旋律轨转换 S_2	仅保留音符数量最多的主旋律轨	多轨复杂音乐
03	截取部分转换 S_3	先筛选主旋律轨(无主轨则选全轨道)，再按预设最大音符数截取前 K 个有效音符	超长或复杂音乐

表 2 中，截取部分转换策略 S_3，需动态计算最大保留音符数 K，计算公式如下：

$$K = \begin{cases} \min\left(500, \left\lfloor \frac{N}{2} \right\rfloor\right) & N > 1000 \\ \min(300, N) & 500 < N \le 1000 \end{cases} \tag{5}$$

式中，N：总音符数

算法采用规则评分体系，将 MIDI 信息的三个输入特征作为三个维度，量化评估三种策略对当前 MIDI 信息的适配程度，为每种策略加分，选择总分最高的策略。评分规则如下：

(1) 有效轨道数量 T：当为单轨($T = 1$)时，S_1 加 3 分；当为多轨($T \ge 2$)时，S_2 加 3 分、S_3 加 2 分；

(2) 总音符数 N：当 $N \le 500$ 时，S_1 加 2 分；当 $500 < N \le 1000$ 时，S_1、S_2 各加 1~2 分；当 $N > 1000$ 时，S_2 加 2 分、S_3 加 3 分；

(3) 主轨占比 R：当 $R > 0.6$ 时，S_2 加 3 分；当 $0.3 < R \le 0.6$ 时，S_2 加 2 分、S_3 加 2 分；当 $R \le 0.3$ 时，S_3 加 3 分。

该算法的决策逻辑为：首先遍历 MIDI 信息所有轨道，提取三个特征值；然后根据评分规则为三种策略累计加分；最后选择得分最高的策略作为输出。若各策略得分相同，按“$S_2 \rightarrow S_3 \rightarrow S_1$”的优先级选择(优先保证旋律完整)。若最优策略为 S_3 (截取部分转换)，则根据总音符数，按式(5)动态计算最大保留音符数 K。算法决策过程的决策树，如图 2 所示。

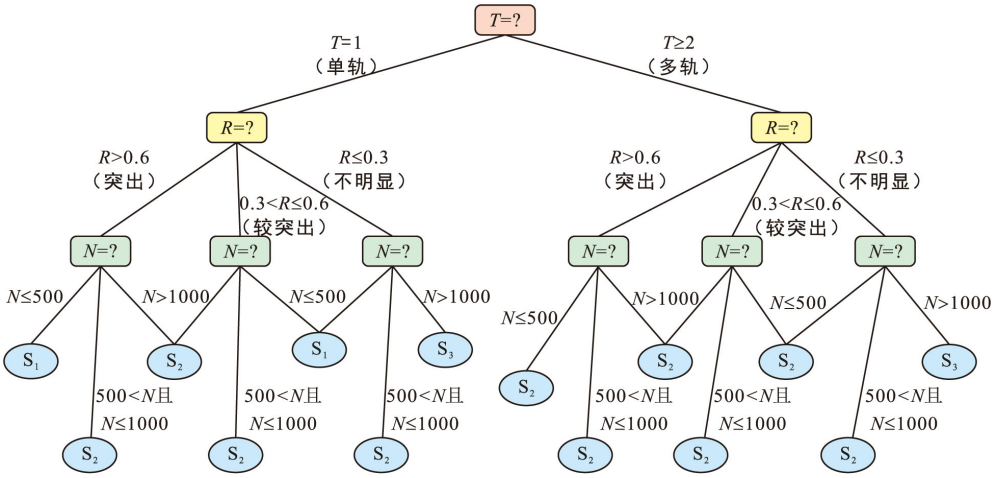


Figure 2. Decision tree for intelligent conversion strategy selection algorithm
图 2. 智能转换策略选择算法决策树

3. MIDI 信息智能转换程序设计与实现

根据第二章的理论模型，本文在 Python 环境下开发了 MIDI 信息转换程序。主函数是整个程序的入口，负责控制整体流程：遍历待转换文件[6]、调用各子函数、处理错误和验证，其执行逻辑如图 3 所示。

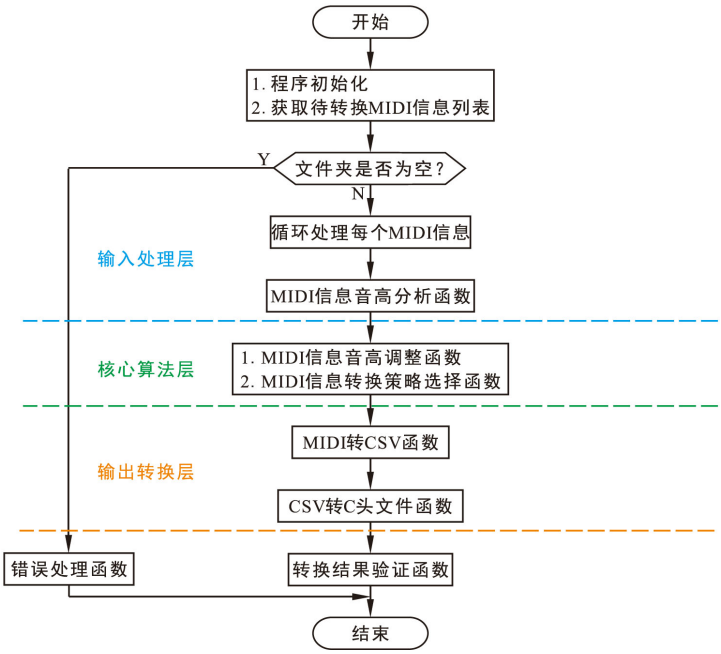


Figure 3. Main function flowchart
图 3. 主函数流程图

图 3 中，程序首先初始化并获取待转换的 MIDI 信息列表；若文件夹为空则触发错误处理；否则循环处理每个文件，依次调用音高分析、音高调整、策略选择、MIDI 转 CSV、CSV 转 C 头文件子函数。所有文件处理完毕后，调用验证函数检查结果，若过程中出现异常则转入错误处理函数。其中，MIDI 信息音高调整函数和 MIDI 信息转换策略选择函数分别为第二章提出的两个核心算法的实现。

3.1. MIDI 信息音高调整函数

MIDI 信息音高调整函数实现了第二章 2.1 节所述的智能音高偏移优化算法。该函数根据输入的音高统计字典，自动计算最佳音高偏移量，将原始 MIDI 信息中的音域适配到木琴有效音域(MIDI 编号 43~67)。算法的具体实现步骤如下：

输入：音高统计字典 pitch_stats (键为 MIDI 编号，值为该音高出现次数)，木琴有效音域 $V = [43, 67]$；

输出：最优音高偏移量 d^* (半音数，取值范围−48~48)。

(1) 初始化 $S^* \leftarrow -\infty$，$d^* \leftarrow 0$；

(2) **for** $d = -48$ to 48 **do**：

a) 初始化 in_range←0，mapped←0；

b) **for each** (p,freq) ϵ pitch_stats **do**：

$p' \leftarrow p + d$；

if p' ϵ V **then** in_range←in_range + freq；

else mapped←mapped + freq；

c) 计算音域内比例 R；

d) 计算音高分布熵 E；

e) 计算中心音高贴近度 C；

f) 计算音域利用率 G；

g) 计算映射惩罚因子 M；

h) 计算位置适应度 P；

i) 计算综合评分 $S \leftarrow 0.25R + 0.20E + 0.15C + 0.15G + 0.10M + 0.15P$；

j) **if** $d \bmod 12 = 0$ **then** $S \leftarrow S \times 1.05$；

k) **if** $S > S^*$ **then** $S^* \leftarrow S$，$d^* \leftarrow d$；

(3) 返回 d^*。

3.2. MIDI 信息转换策略选择函数

MIDI 信息策略选择函数实现了第二章 2.2 节所述的智能转换策略选择算法。该函数根据 MIDI 信息的轨道结构特征，自动选择最优转换策略，以适应不同的音乐类型和系统资源限制。算法的具体实现步骤如下：

输入：MIDI 信息路径；

输出：策略代码 strategy ϵ {1, 2, 3}，最大保留音符数 K (仅当 strategy = 3 时有效)。

(1) 读取 MIDI 信息，遍历所有轨道，统计：有效轨道数 T、总音符数 N 及主轨占比 R；

(2) 初始化三种策略得分 $score_1 = score_2 = score_3 = 0$；

(3) 根据有效轨道数加分：

if $T = 1$ **then** $score_1 += 3$；

else $score_2 += 3$，$score_3 += 2$；

(4) 根据总音符数加分：

if $N \leq 500$ **then** $score_1 += 2$；

else if $500 < N \leq 1000$ **then** $score_1 += 1$，$score_2 += 2$；

else $score_2 += 2$，$score_3 += 3$；

(5) 根据主轨占比加分：

if $R > 0.6$ **then** $score_2 += 3$；

else if $0.3 < R \leq 0.6$ **then** $score_2 += 2$，$score_3 += 2$；

else $score_3 += 3$；

(6) 选择得分最高的策略：strategy = argmax ($score_1$, $score_2$, $score_3$)；若平分，按 $S_2 \rightarrow S_3 \rightarrow S_1$ 优先级选择；

(7) **if** strategy = 3 **then**：

if $N > 1000$ **then** $K = \min(500, \lfloor N/2 \rfloor)$；

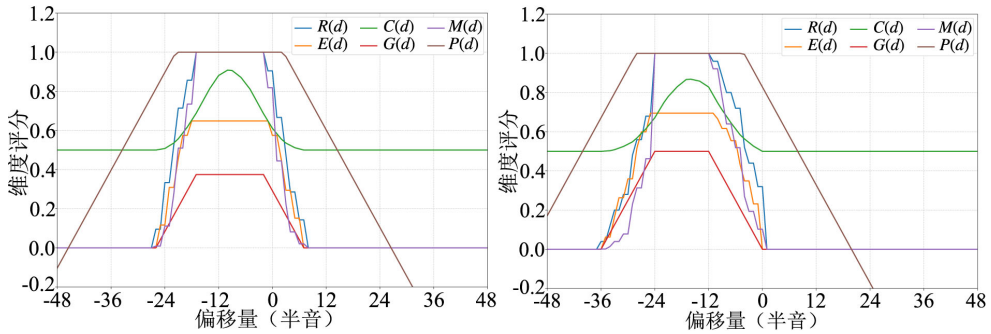
else if $500 < N \leq 1000$ **then** $K = \min(300, N)$；

(8) 返回 strategy，K。

4. 核心算法验证与分析

4.1. 智能音高偏移优化算法的验证

以《Little Star》和《Happy Birthday》为例，在 Python 环境下，对两个 MIDI 文件进行音高偏移量穷举搜索(范围−48 至 48 半音)，各维度评分随偏移量变化的曲线，如图 4 所示。

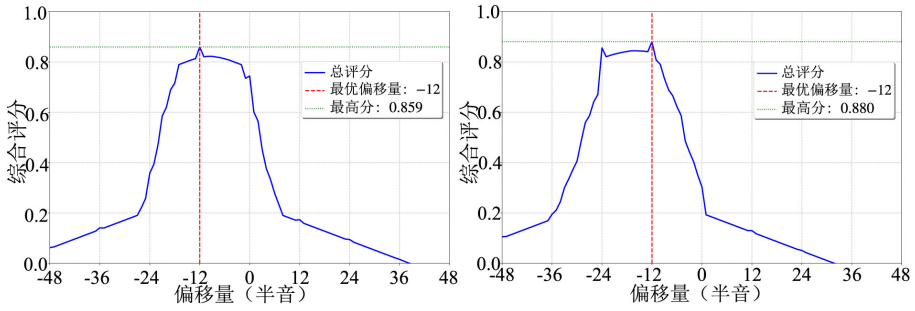


(a) 《Little Star》MIDI 文件　　(b) 《Happy Birthday》MIDI 文件

Figure 4. Dimensional scoring curves of two MIDI files
图 4. 两个 MIDI 文件维度评分曲线图

图 4 中，据第二章表 1 分析可知，各评分函数值越接近 1 表示该维度表现越优。因此，《Little Star》MIDI 文件在原始音高(0 半音)或轻微降调(−12 半音)时，整体音乐表现最佳；《Happy Birthday》MIDI 文件在轻微降调(−12 半音)或−24 半音时，整体音乐表现最佳。

综合评分随偏移量变化的曲线，如图 5 所示。



(a) 《Little Star》MIDI 文件　　(b) 《Happy Birthday》MIDI 文件

Figure 5. Comprehensive scoring curve of two MIDI files
图 5. 两个 MIDI 文件综合评分曲线图

图 5 表明，两个 MIDI 文件均在偏移量为−12 半音时取得最高综合评分，证明算法能有效将音域适配至木琴有效范围。

4.2. 智能转换策略选择算法的验证

以《Little Star》和《Happy Birthday》为例，在 Python 环境下，算法首先提取两个 MIDI 文件的三个关键特征，如表 3 所示。

Table 3. Three key characteristics of two MIDI files
表 3. 两个 MIDI 文件的三个关键特征

NO.	歌曲名称	有效轨道数量 T	总音符数 N	主轨占比 R
01	《Little Star》	1	42	1.00
02	《Happy Birthday》	1	356	1.00

根据第二章的评分规则计算三种策略的综合得分，结果如表 4 和表 5 所示。

Table 4. Scoring calculation table for intelligent conversion strategies of “Little Star”
表 4. 《Little Star》智能转换策略评分计算表

NO.	评分维度	特征区间	策略加分			选择结果
			S_1	S_2	S_3	
01	有效轨道数量	T = 1 (单轨)	+3	+1	+0	
02	总音符数	N = 42 (少)	+2	+1	+0	
03	主轨占比	R = 1.00 (突出)	+1	+3	+1	
总计			6	5	1	策略 1 (S_1)

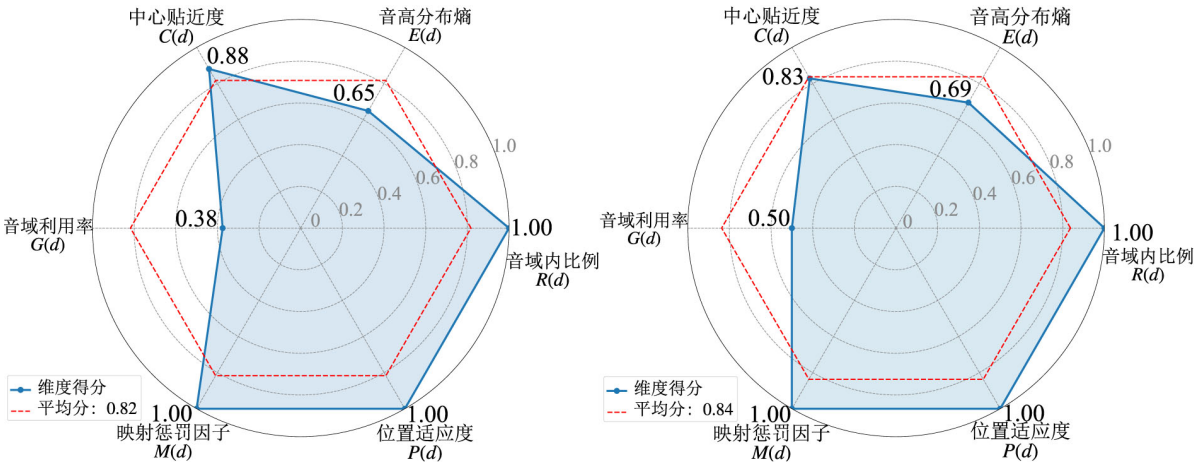
Table 5. Calculation table for the score of the intelligent conversion strategy of “Happy Birthday”
表 5. 《Happy Birthday》智能转换策略评分计算表

NO.	评分维度	特征区间	策略加分			选择结果
			S_1	S_2	S_3	
01	有效轨道数量	T = 1 (单轨)	+3	+1	+0	
02	总音符数	N = 356 (少)	+2	+1	+0	
03	主轨占比	R = 1.00 (突出)	+1	+3	+1	
总计			6	5	1	策略 1 (S_1)

两个案例中策略 1 (全曲转换)得分最高，与单轨简单音乐的特征相符，验证了算法决策的正确性。

4.3. 算法综合评估

为直观展示算法效果，以《Little Star》和《Happy Birthday》为例，对两个 MIDI 文件最优音高偏移量得分进行可视化分析，如图 6 所示。



(a) 《Little Star》MIDI 文件　　(b) 《Happy Birthday》MIDI 文件

Figure 6. Radar chart of the optimal pitch offsets for two MIDI files
图 6. 两个 MIDI 文件最优音高偏移量雷达图

图 6 中，雷达图展示了两个 MIDI 文件在最优偏移量下的六个维度得分，多边形面积越大表示综合表现越好，说明智能音高偏移优化算法能识别音乐的类型与特征，据音乐特性进行智能优化。

在 Python 环境下，对多个 MIDI 文件运用智能转换策略选择算法选择最优转换策略，得到的最优转换策略选择统计柱状图，如图 7 所示。

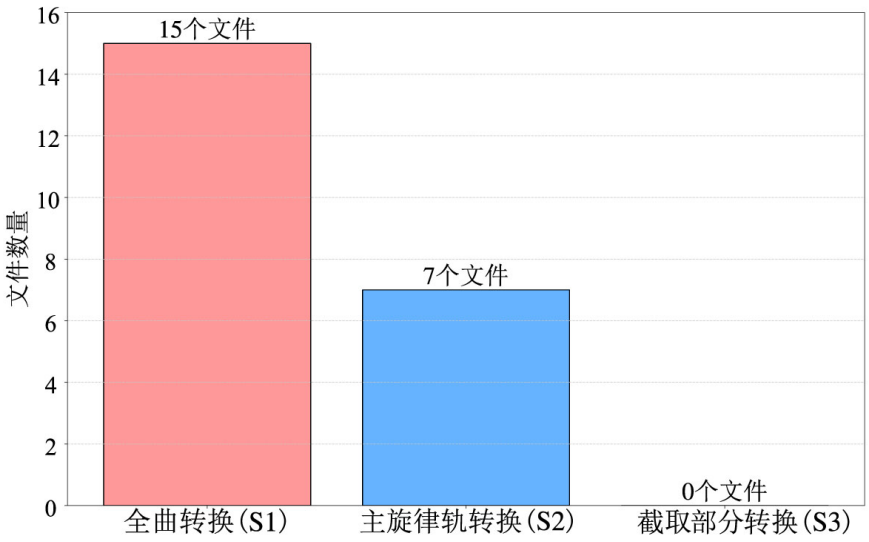
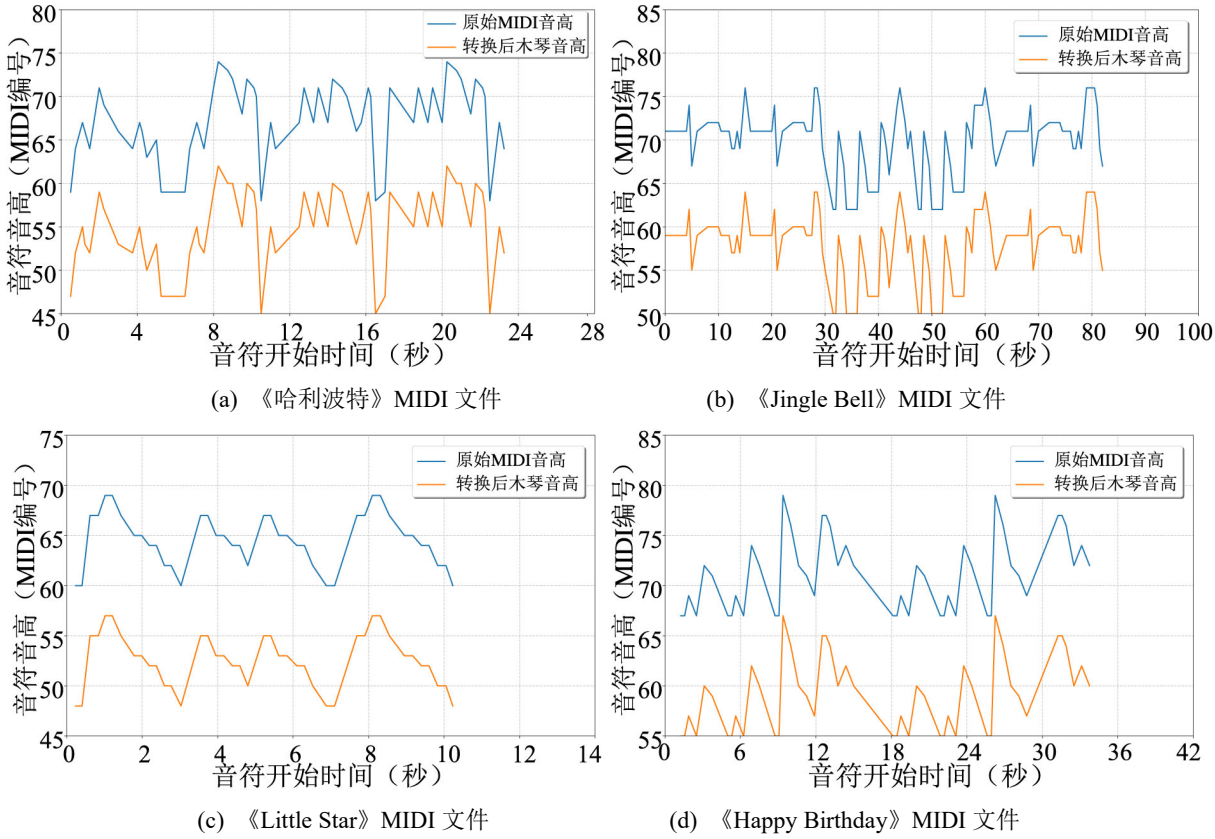


Figure 7. Statistical bar chart of optimal conversion strategy selection
图 7. 最优转换策略选择统计柱状图

由图 7 可知，智能转换策略选择算法在大多数情况下倾向于选择全曲转换策略(尽量不选截取部分转换策略)，以保证歌曲旋律的完整性。同时，少部分情况选择主旋律轨转换策略，说明算法可据音乐内容动态选择策略。

在 Python 环境下，运行 MIDI 信息转换程序，在多个算法支持下得到的转换前后四个 MIDI 文件的音高分布折线图，如图 8 所示。



(a) 《哈利波特》MIDI 文件　(b) 《Jingle Bell》MIDI 文件

(c) 《Little Star》MIDI 文件　(d) 《Happy Birthday》MIDI 文件

Figure 8. Line graph showing the pitch distribution of four MIDI files
图 8. 四个 MIDI 文件音高分布折线图

由图 8 可知，四个 MIDI 文件转换后的音高全部落在十五音木琴音域内(MIDI 编号 43~67)，且转换前后折线形状大致相似。表明程序能够正确适配不同音高范围和时长的 MIDI 文件，完整保留原始旋律轮廓与时间节奏。进一步分析两种算法的特性：智能音高偏移优化算法通过多维度加权评分全面评估音高分布，能找到较优偏移量、保持音乐质量，但计算开销较大且存在局部最优风险，适用于对质量要求高的关键曲目；智能转换策略选择算法简单高效、决策快速，适合实时应用，但特征维度较单一，可能误选策略，适用于大量 MIDI 信息的初步筛选。

5. 结论

本文针对电子木琴自动演奏系统中的 MIDI 信息智能转换问题，提出了两种核心算法，并进行了系统性的实现与验证。智能音高偏移优化算法通过六维度加权评分体系，自动计算最优音高偏移量，有效解决了有限音域乐器与宽音域 MIDI 信息之间的音域适配问题。智能转换策略选择算法基于轨道数、总音符数和主轨占比三个特征，通过规则评分自动选择最优转换策略，兼顾了旋律完整性与系统资源限制。在 Python 环境下开发的转换程序，实现了从 MIDI 信息到 C 语言头文件的完整转换流程。实验结果表明，两种算法均有效，转换后的音高分布与原始旋律轮廓高度一致，验证了算法的正确性和适用性。

参考文献

[1] 郭凌华, 刘明磊, 丁亭文, 等. 基于 LabVIEW 数字图像生成音乐旋律算法的研究[J]. 西安理工大学学报, 2020, 36(1): 65-71.

[2] 谢林玲. 多音轨 MIDI 音乐主旋律提取方法探析[J]. 科技创新导报, 2021, 18(19): 72-74.

[3] 刘嘉欣. 嵌入式 MIDI 文件格式解析设计与实现[J]. 微计算机信息, 2006(32): 66-68+42.

[4] 师冬冬. 基于深度学习的 MIDI 信号对电子音乐旋律自动转换研究[J]. 自动化与仪器仪表, 2024(11): 20-23+28.

[5] 杨世瑞, 程科, 李想, 等. 基于多音轨聚类的 MIDI 音乐主旋律提取方法[J]. 计算机与数字工程, 2024, 52(10): 2897-2901.

[6] 严桂林, 陈学煌, 赵云娥. 基于 S3C44B0 的 MIDI 音乐发生器的设计与实现[J]. 电子测量技术, 2008(8): 158-161.

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(6), 375-390
Published Online December 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.26037

基于AI辅助工程的混合MCU－边缘－云平台的工业AIoT系统设计与实现

吴　薇[1,2]

[1]杭州电子科技大学电子信息学院，浙江　杭州
[2]江苏矽望电子科技有限公司，江苏　南京

收稿日期：2026年4月20日；录用日期：2026年5月1日；发布日期：2026年5月27日

摘　要

文章提出一种面向混合微控制器－边缘－云端架构工业AIoT系统的嵌入式AI辅助工程化方法(AI-Assisted Engineering for embedded systems)，用于支持嵌入式软件、RTOS定制、传感器信息模型、通信协议、边缘轻量级学习算法、云端协议及数据挖掘的协同开发，以区别于仅依赖提示生成代码的“氛围编程”(Vibe Coding)。文章以“项目速度”(Project Velocity)原型系统(STM32、RK3588与ThingsBoard)为参考实现，展示了从裸硬件起步、在AI辅助工程化方法下完成端到端软件系统构建的全过程。该架构将信号采集、预处理及故障安全行为部署于微控制器传感器节点，将协议转换、上下文融合、本地推理、数据缓冲、轻量级机器学习及人机交互功能部署于边缘网关，将数据分析、模型注册、受控空中升级(OTA)及运维管理部署于云端。针对嵌入式开发中“AI提示优先”方式易引发的硬件映射缺失、时序约束不清、跨层配置不一致及不安全输出等问题，文章提出面向Zephyr RTOS的“三文件”提示策略，要求AI同时生成硬件覆盖文件(.overlay/.dts)、项目配置文件(prj.conf)和主程序文件(main.c)，从而建立硬件定义、系统配置与应用逻辑之间的闭环。进一步地，文章构建了覆盖需求、实现、验证、文档与部署的规范化工作流程，并结合人工审查、回归测试、硬件在环验证、安全扫描和受控滚动发布等机制，提高系统的可验证性、可维护性与部署可靠性。最后，结合预测性维护用例与多维评价框架，讨论了该方法在实际部署及后续规模化验证中的应用价值。

关键词

工业AIoT，AI辅助工程，边缘计算，Zephyr实时操作系统，负责任AI，系统可追溯性

AI-Assisted Engineering for Developing a Hybrid MCU-Edge-Cloud Industrial AIoT System

Wei Wu[1,2]

[1]School of Electronics and Information Engineering, Hangzhou Dianzi University, Hangzhou Zhejiang
[2]Cynoware Electronics, Inc., Nanjing Jiangsu

文章引用: 吴薇. 基于 AI 辅助工程的混合 MCU-边缘-云平台的工业 AIoT 系统设计与实现[J]. 嵌入式技术与智能系统, 2025, 2(6): 375-390. DOI: 10.12677/etis.2025.26037

Received: April 20, 2026; accepted: May 1, 2026; published: May 27, 2026

Abstract

This paper proposes AI-Assisted Engineering, a structured AI-supported implementation methodology for industrial AIoT systems built on a hybrid microcontroller-edge-cloud architecture. The method supports the coordinated development of embedded software, RTOS customization, sensor information models, communication protocols, lightweight edge learning algorithms, cloud protocols, and data mining, and is intended to address the limitations of prompt-first "vibe coding." Using the Project Velocity prototype system (STM32, RK3588, and ThingsBoard) as a reference implementation, the paper presents an end-to-end software development process that starts from bare hardware and is completed with AI assistance. In the proposed architecture, signal acquisition, preprocessing, and fail-safe behaviors are assigned to microcontroller-based sensor nodes; protocol translation, context fusion, local inference, data buffering, lightweight machine learning, and human-machine interaction are handled by the edge gateway; while data analytics, model registration, controlled over-the-air (OTA) updates, and system management are deployed in the cloud. To mitigate common issues in AI-prompt-driven embedded development—particularly missing hardware mappings, unclear timing constraints, cross-layer inconsistencies, and unsafe outputs—the paper introduces a three-file prompting strategy for Zephyr RTOS, requiring AI-generated deliverables to include a hardware overlay file (.overlay/.dts), a project configuration file (prj.conf), and an application source file (main.c). This strategy establishes a closed loop between hardware definition, system configuration, and application logic. The paper further presents a standardized workflow covering requirements, implementation, verification, documentation, and deployment, reinforced by human review, regression testing, hardware-in-the-loop validation, security scanning, and controlled rolling release. Finally, a predictive maintenance use case and a multi-dimensional evaluation framework are discussed to support practical deployment and future large-scale validation.

Keywords

Industrial AIoT, AI-Assisted Engineering, Edge Computing, Zephyr RTOS, Responsible AI, Traceability

1. 引言

随着工业互联网、边缘计算与嵌入式人工智能技术的持续融合，工业现场系统正由传统的单点感知与数据上传模式，逐步演进为集感知、推理、执行、回传与优化于一体的动态闭环体系[1] [2]。在这一演进过程中，工业系统的智能化不再仅仅体现为云端算法能力的增强，而是越来越依赖于设备侧、边缘侧与云侧之间的协同设计与分层实现。相较于传统以云平台为中心的系统架构，面向工业人工智能物联网(Industrial AIoT, AIoT)的系统构建不仅需要综合考虑算力配置、接口连接与数据流转，还必须系统性地纳入实时响应、故障可恢复、弱网运行、安全可控、版本可治理以及工程可追溯等多重约束。尤其是在设备节点、现场总线、边缘网关、云平台及运维流程深度耦合的复杂应用场景中，系统开发的核心挑战已不再局限于局部算法优化或单板级功能实现，而是演变为贯穿硬件定义、实时软件配置、协议协同、跨层数据一致性、验证测试与部署治理等多个环节的系统工程问题。对于这类混合微控制器–边缘–云端

架构的工业 AIoT 系统，单纯依赖代码生成或模块拼接的开发方式，往往难以满足工程落地对可靠性、可维护性和可验证性的要求。

氛围编程(Vibe Coding)，是指一种以提示驱动为核心、强调根据高层意图快速生成代码的开发方式，但通常缺乏对硬件映射、时序约束、跨层一致性及系统验证的显式处理。面向嵌入式系统的 AI 辅助工程(AI-Assisted Engineering)，是指一种结构化工程方法，其中 AI 在嵌入式系统全生命周期中辅助实现相关工作，但始终受到人工定义的架构、硬件、时序、安全与验证约束，其目标不是生成孤立代码，而是产出可构建、可测试、可部署的系统。

Project Velocity (项目速度)是一个面向嵌入式系统的 AI 辅助工程原型项目，基于 STM32 传感器节点、RK3588 边缘网关和 ThingsBoard 云平台，从裸硬件出发构建面向工业 AIoT 的混合 MCU－边缘－云端可部署系统。

本文的主要贡献如下

(1) 提出一种面向嵌入式系统的 AI 辅助工程方法，将 AI 系统化引入需求分解、硬件映射、RTOS 配置、驱动与协议实现、测试验证、文档整理以及部署维护等开发环节，并在人工主导的架构、时序、安全与验证约束下，形成适用于工业 AIoT 系统的结构化工程流程。

(2) 将面向工业 AIoT 的混合微控制器－边缘－云端系统架构，依据时序敏感性、推理实时性与治理需求进行层次化任务划分：将信号采集与故障安全下沉至 MCU 节点，将协议转换、上下文融合、本地推理与轻量级学习部署于边缘侧，并将数据分析、模型注册、OTA 管理与运维治理部署于云端，同时明确运营技术(OT)与信息技术(IT)的接口边界。

(3) 针对嵌入式项目中 AI 生成代码易出现的硬件映射缺失、配置不一致与逻辑孤立等问题，提出面向 Zephyr 等框架的“三文件”提示策略与上下文注入方法，要求 AI 协同生成硬件覆盖文件(.overlay/.dts)、项目配置文件(prj.conf)和主程序文件(main.c)，从而增强工程项目的可构建性、可调试性与跨层一致性。

(4) 构建覆盖需求、实现、验证、文档与部署全过程的质量保障机制，将人工审查、回归测试、硬件在环验证、安全扫描及受控滚动发布纳入开发流程，并结合预测性维护典型用例与多维评估框架，为后续量化验证与规模化部署提供可操作的方法基础和指标支撑。

2. 氛围编程和面向嵌入式系统的 AI 辅助工程

随着大模型与代码生成技术的发展，基于自然语言提示快速生成程序的开发模式受到广泛关注。其中，所谓“氛围编程(Vibe Coding)”，通常是指一种以提示驱动为核心、强调依据高层意图快速生成代码的开发方式。该方式在原型验证、界面搭建及通用软件场景中具有一定效率优势，但在嵌入式系统开发中，往往缺乏对硬件映射、时序约束、系统配置依赖、跨层一致性与安全验证的显式处理，因而容易产生“代码表面可行而系统整体不可用”的问题。特别是在 RTOS、驱动、中断、总线协议及故障安全机制密切耦合的场景下，这种提示优先、逻辑优先的开发方式存在明显局限。

为应对上述问题，本文提出一种面向嵌入式系统的 AI 辅助工程(AI-Assisted Engineering)方法。该方法并非将 AI 视为替代工程师的自动编码工具，而是将其作为工程协同助手，引入到需求分析、硬件映射、RTOS 配置、驱动与协议实现、测试验证、文档生成及部署维护等开发环节之中，并始终置于人工定义的架构、硬件、时序、安全与验证约束之下。其目标不是生成孤立的代码片段，而是形成可构建、可测试、可部署、可演进的嵌入式系统工程产物。

嵌入式系统中的 AI 辅助工程(AI-Assisted Engineering)，是指在明确工程约束和人工主导验证的前提下，将人工智能作为辅助工具的一种系统化工程方法。其核心目标不是替代工程师，而是提升开发速度、覆盖范围与知识复用效率，同时保证嵌入式系统在时序、资源、接口、安全性、可靠性和可维护性等方面满足工程要求。下面的表 1 是“氛围编程”与面向嵌入式系统的 AI 辅助工程的对比。

Table 1. Comparison between “Vibe Coding” and AI-assisted engineering for embedded systems
表 1. “氛围编程”与面向嵌入式系统的 AI 辅助工程的对比

对比维度	面向嵌入式系统的“氛围编程”	面向嵌入式系统的 AI 辅助工程
主要目标	根据自然语言意图快速生成代码	在真实工程约束下交付可运行、可验证的嵌入式系统
开发方式	提示优先、代码优先	需求优先、架构优先、验证驱动
对硬件的看法	往往被抽象化或默认假设存在	明确建模：引脚、总线、时钟、中断、电源、内存、外设等
AI 的典型输出	孤立代码片段，或“看起来正确”的逻辑	协同交付物：硬件配置、RTOS/项目配置、源代码、测试、文档等
系统上下文感知能力	通常较弱	强调全栈上下文：MCU、RTOS、协议、边缘、云端、部署
对约束的处理	常忽略时序、RAM/Flash 限制、并发、安全状态等	将时序、内存、功耗、安全、接口约束作为一等设计输入
RTOS 集成能力	可能生成任务或逻辑，但缺少有效配置与依赖	能对齐任务、优先级、Kconfig/prj.conf、设备树/overlay、驱动与中间件
软硬件一致性	软硬件一致性	显式保证一致性
验证方式	“先跑跑看”	人工审查、构建检查、回归测试、硬件在环、协议测试、安全检查
输出结果的可靠性	速度快，但在真实产品中稳定性差	前期更严谨，但更适合从原型走向产品
人的角色	事后审查生成结果	担任架构师、约束定义者、验证者和发布责任人
最适用场景	早期创意验证、原型草图、快速实验	真实嵌入式开发、工业物联网、涉及安全的功能、可部署系统

本文以 Project Velocity (项目速度)为原型参考实现，构建了一个基于 STM32 传感器节点、RK3588 边缘网关和 ThingsBoard 云平台的混合 MCU－边缘－云端工业 AIoT 系统[3]，其中轻量级 AI 算法 TinyML [3] [4]处于 STM32 或传统 AI 算法处于 RK3588 中，如图 1 所示。

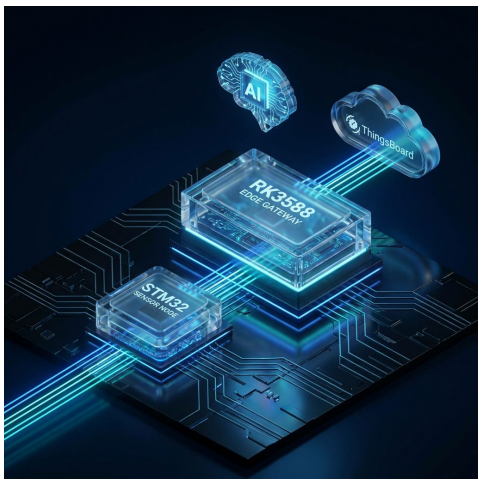


Figure 1. Composition of the “Project Velocity” prototype system
图 1. “项目速度”(Project Velocity)原型系统组成

项目从裸硬件出发、借助本文面向嵌入式系统的 AI 辅助工程(AI-Assisted Engineering)方法完成端到端软件系统构建。在该系统中，信号采集、预处理及故障安全行为部署于微控制器侧；协议转换、上下文融合、本地推理、数据缓冲、轻量级学习算法及人机交互功能部署于边缘侧；数据分析、模型管理、受控空中升级(OTA)与运维治理功能部署于云端。针对嵌入式项目中 AI 生成内容常见的硬件定义缺失、配置文件不匹配、逻辑与工程上下文脱节等问题。

3. 系统需求与总体架构

3.1. 设计目标与分层原则

工业 AIoT 系统设计的核心挑战在于，其各层级在功能、性能与约束上存在根本性差异。底层 MCU 直接对接物理世界的传感器、执行器与现场总线，其设计首要遵循确定性时序、低功耗与故障安全原则；边缘网关承载上下文融合、协议转换、本地推理及现场人机交互，重点关注数据吞吐、缓存能力、断网续传与接口兼容性；而云平台则负责舰队级监控、模型迭代、受控 OTA 升级与审计追溯，强调全局可视、策略一致性与长期治理。因此，系统架构必须在设计初期就清晰划分决策边界——明确“哪些控制回路必须在本地实时闭环内完成，哪些功能可上移至边缘或云端协同”，从而避免因职责模糊导致的时延失控、可靠性下降与运维责任不清。

3.2. 混合 MCU－边缘－云总体架构

“项目速度”采用分层协同的 MCU (STM32)－边缘(RK3588)－云(ThingsBoard)混合架构，如图 2 所示。

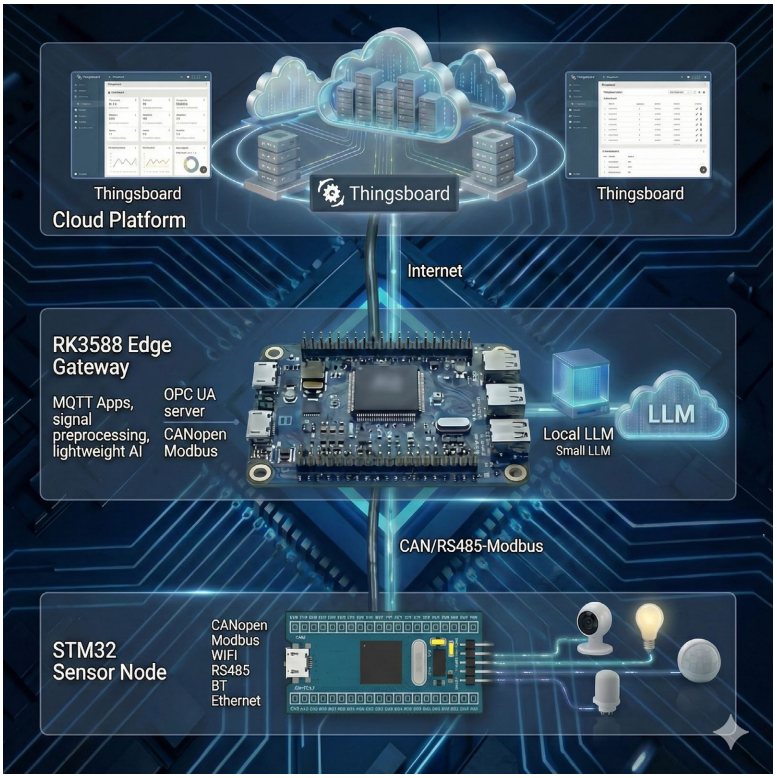


Figure 2. MCU (STM32) - edge (RK3588) - cloud (ThingsBoard) hybrid architecture
图 2. MCU (STM32)－边缘(RK3588)－云(ThingsBoard)混合架构

3.2.1. STM32 传感器/执行节点

在 STM32 传感器/执行节点层，系统主要承担现场感知、状态输入与执行控制等功能。感知与输入部分包括 DHT11/22 温湿度传感器、LM35 温度传感器、热敏电阻分压采样电路、红外接收模块、4 × 4 矩阵键盘、RFID-RC522 射频识别模块以及用于门锁状态检测的磁簧开关等；执行与输出部分包括直流电机驱动模块、继电器控制的电磁锁、I2C 接口 16 × 2 LCD 显示模块以及红外发射电路等。其中，红外发射器与红外接收器共同用于实现对 Lasko 加热器的红外遥控与指令学习功能。节点与上位边缘网关之间通过 UART 串行接口进行通信，并通过 CAN 总线及 CAN 收发器扩展现场总线接入能力；同时，系统还保留了调试与下载接口，用于固件烧录、在线调试与系统维护。整体硬件连接关系如图 3 所示。

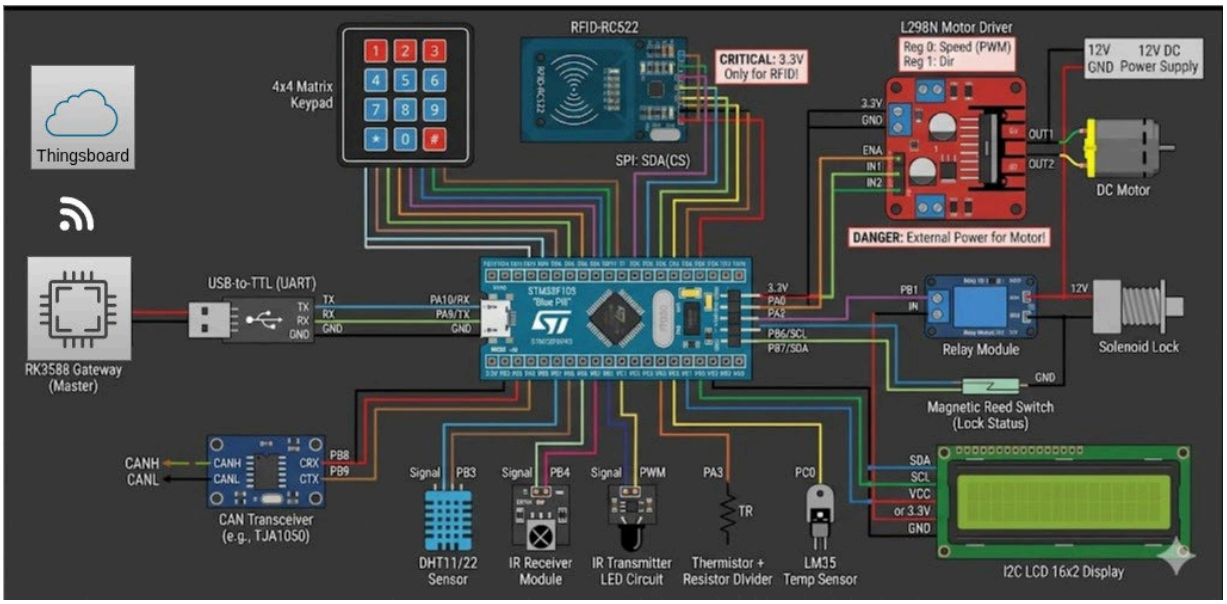


Figure 3. STM32 sensor and actuator node module
图 3. STM32 传感器和执行节点模块

在 STM32 传感器/执行节点的软件平台选择上，本文采用工业级 ZephyrRTOS [5]，而非简单裸机程序或轻量级单任务框架，主要基于实时性、模块化、可扩展性以及后续智能化能力演进等多方面考虑。工业 AIoT 现场节点通常需要同时承担多源传感器采集、执行器控制、周期性任务调度、通信协议处理及异常状态响应等功能，对系统的确定性、并发处理能力和工程可维护性提出了较高要求。ZephyrRTOS 提供线程调度、定时器、中断管理、消息队列、信号量及事件机制，能够较好支持温湿度采集、矩阵键盘扫描、红外收发、LCD 刷新、门锁控制以及与边缘网关的数据通信等任务的协同运行。与此同时，Zephyr 基于 Devicetree、Kconfig 和模块化驱动框架的工程组织方式，可将硬件定义、系统配置与应用逻辑有效解耦，从而提高软件系统的可移植性、可维护性和可测试性，这对于本文所提出的 AI 辅助工程方法尤为关键，因为其能够为 AI 生成结果提供清晰的硬件映射边界和配置约束，减少“逻辑可生成但工程不可构建”的问题。

进一步地，Zephyr RTOS 与 TinyML 在资源受限嵌入式平台上的设计目标具有较高一致性，因此非常适合作为 STM32 节点侧轻量级智能推理的运行基础。TinyML 模型通常强调小参数规模、低功耗、低内存占用和本地实时推理，适合部署在 Cortex-M 等微控制器平台；而 Zephyr 所提供的轻量化内核、可裁剪组件机制、标准化驱动接口与实时任务调度能力，能够为 TinyML 推理任务提供稳定、可控的运行

环境。在实际系统中，模型推理可作为独立线程、定时触发任务或事件响应流程嵌入节点应用，与传感器采样、特征提取、阈值判断和执行控制形成闭环联动，从而实现更低时延、更强鲁棒性的本地智能处理能力。此外，Zephyr 在 UART、I2C、SPI、GPIO、PWM、CAN 等接口上的统一抽象，也便于将 TinyML 与现场感知、通信和控制链路进行集成，降低模型落地的工程复杂度。因而，从实时性、轻量化、组件化配置、外设协同以及本地智能部署能力等方面综合考虑，Zephyr RTOS 不仅是 STM32 工业节点实现的合适基础软件平台，也为 TinyML 在节点侧的工程化部署提供了良好的支撑。

3.2.2. RK3588 边缘网关

在边缘网关层，本文选用 RK3588 作为核心处理平台，主要基于以下考虑。首先，RK3588 属于高性能边缘 AI SoC，集成八核 64 位 CPU 与 6 TOPS NPU，并提供 PCIe、USB 3.0、SATA、双千兆以太网、UART、SPI、I2C 等丰富高速与低速接口，能够同时支撑传统工业协议处理、本地规则引擎、多源数据融合、视频/图像类算法扩展以及边缘侧智能推理等任务，较适合作为工业 AIoT 系统中的边缘计算与协议汇聚节点。其次，RK3588 及其生态已广泛面向机器人、机器视觉、工业控制、边缘计算等场景展开应用，Rockchip 官方资料也明确将 RK3588 视为新一代机器人方案的重要平台之一，这说明其在性能、接口能力与产业生态方面具备较好的工程基础。再次，从软件平台角度看，基于 RK3588 的主流开发板生态已提供对 Ubuntu 22.04 等 Linux 发行版的支持，例如 Ubuntu 22.04 作为长期可运行系统之一，这为长期维护、依赖管理、容器化部署、远程运维以及后续版本演进提供了较成熟的软件基础。对于本文所实现的边缘网关而言，Ubuntu LTS 体系还便于稳定承载 Python、Docker、OPC UA [6]、MQTT [7]、数据库、可视化和 AI 推理相关软件栈，从而降低系统集成与长期维护成本。

基于上述硬件与软件基础，在边缘网关的 RK3588 层，系统运行 Linux/Ubuntu 平台软件栈，实现 OPC UA、CANopen [8]/Modbus 与 MQTT 等异构工业协议之间的实时桥接，并融合多源上下文信息完成本地异常检测、告警生成、数据缓存与规则执行等功能；在云端平台层，则统一承担模型版本管理、远程策略下发、多维可视化分析、系统状态归档以及受控 OTA 更新等职责。该分层设计遵循“越靠近物理层，时序确定性与故障安全要求越高；越靠近系统上层，治理覆盖范围与全局协同能力越强”的原则：MCU 节点负责强实时感知与执行闭环，RK3588 边缘网关负责协议汇聚、局部智能与弱网自治，而云端侧则负责全局分析、模型治理与系统运维。这样的层级划分既发挥了 RK3588 在边缘算力、接口扩展与 Linux 软件生态方面的优势，也有助于提升工业 AIoT 系统的实时性、鲁棒性与可维护性，如图 4 所示。

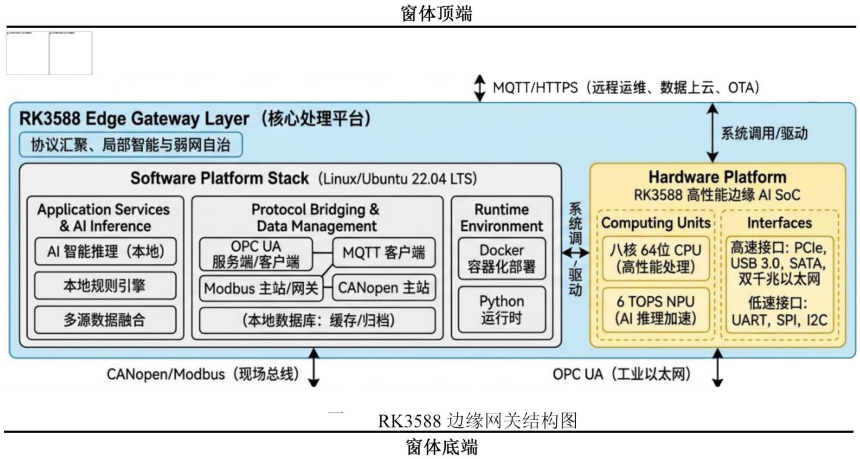


Figure 4. RK3588 edge gateway architecture diagram
图 4. RK3588 边缘网关结构图

3.2.3. 云端 ThingsBoard

在云平台层，本文采用 ThingBoard 作为工业 AIoT 系统的统一接入与可视化管理平台，用于实现设备遥测数据接收、远程控制、人机交互展示、规则联动以及历史数据管理等功能。如图 5 所示，系统在 ThingBoard 仪表板中集成了多类现场设备与控制对象的状态监测和远程操作能力，包括 Lasko 加热器红外遥控、直流电机转速与方向控制、LCD 显示内容下发、矩阵键盘输入状态显示、LED 开关控制以及温度等传感器数据的实时可视化。通过这些组件，云平台能够将现场 MCU 节点和边缘网关上传的数据以统一界面进行展示，并支持操作人员通过网页端下发远程控制指令，实现从“状态感知”到“控制执行”的闭环交互。

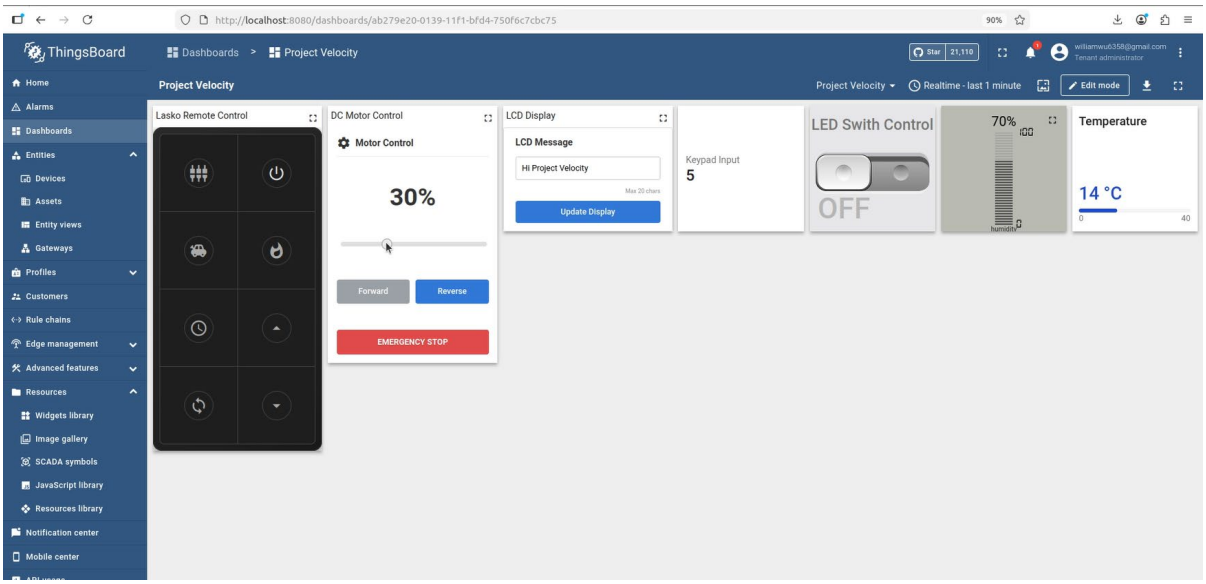


Figure 5. Cloud ThingsBoard dashboard
图 5. 云端 ThingBoard 仪表板

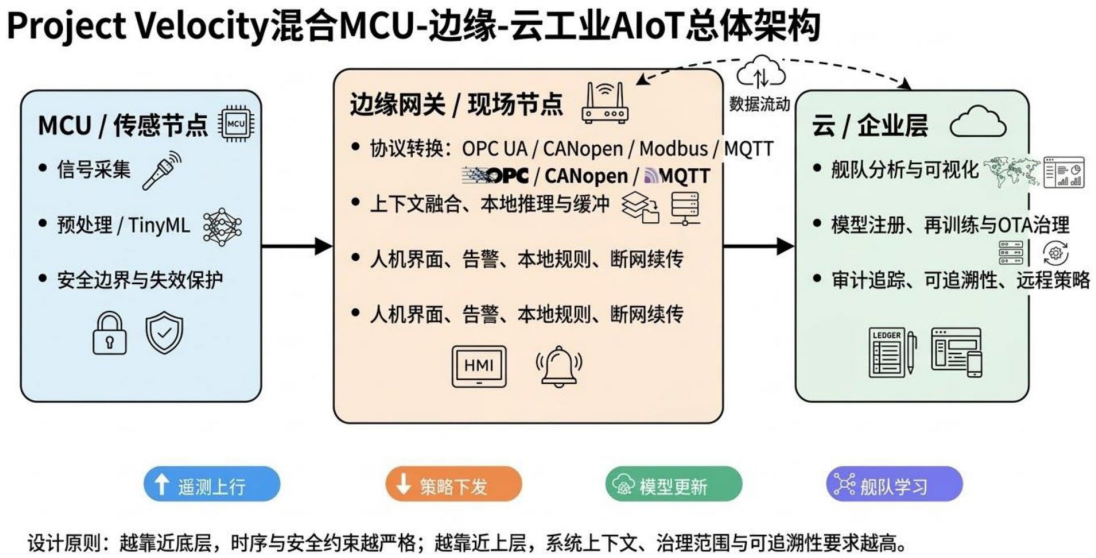


Figure 6. Overall hybrid MCU-Edge-cloud industrial AIoT architecture of “Project Velocity”
图 6. “项目速度”混合 MCU－边缘－云工业 AIoT 总体架构

在系统实现上，ThingBoard 主要承担以下几类功能：其一，作为设备遥测与属性数据的统一入口，接收来自 RK3588 边缘网关经 MQTT 上传的实时数据，并完成存储、展示与历史归档；其二，作为远程控制与运维交互界面，通过 RPC 或命令下发机制，将用户在仪表板上的控制动作转发至边缘网关及 STM32 执行节点，实现电机调速、显示更新、设备开关控制等操作；其三，作为规则与告警管理平台，对温度、状态变化及异常条件进行联动处理，为后续异常告警、日志审计和运维决策提供支撑；其四，作为系统治理与扩展接口，为后续接入模型版本管理、远程策略配置、OTA 升级和多设备集中管理提供统一云端基础。

从架构分层角度看，ThingBoard 云平台并不承担强实时控制闭环，而是侧重于全局可视化、远程访问、历史分析和集中治理。通过将实时性要求较高的感知与执行功能保留在 STM32 节点及 RK3588 边缘层，而将多终端访问、运维管理、可视化分析和策略下发部署于云平台层，系统能够在兼顾现场响应能力的同时，提高整体系统的可管理性、可扩展性和工程部署效率。

3.3. 端到端数据与决策闭环

基于上述分层架构，系统端到端的数据流与决策闭环可抽象为六个相互衔接的阶段：感知、预处理、推理、解释与告警、执行与留痕、再训练与更新。只有当每个阶段之间的交接点均完整保留版本信息、告警依据、现场上下文与执行记录时，系统才具备真正可验证的可信度。换言之，工业 AIoT 系统的关键不仅在于单点识别精度，更在于整个跨层链路在面对真实故障、网络中断与版本迭代时，所保持的全程可追溯性与一致性。

4. AI 辅助工程方法

4.1. 嵌入式与 AIoT 中的“氛围编程”局限性

在简单脚本开发、网页原型构建或纯云端服务场景中，“氛围编程”能够凭借提示驱动快速生成可运行的代码原型，因而具有较高的试错效率和开发速度。然而，在嵌入式系统与工业 AIoT 场景中，其局限性更加明显，且往往直接影响系统的可交付性、可靠性与安全性。

首先，嵌入式开发高度依赖具体硬件平台，AI 生成代码时容易忽略底层硬件的关键约束，例如引脚复用关系、时钟树配置、外设初始化顺序、DMA 通道占用、中断优先级、看门狗策略以及低功耗状态切换等。这些内容通常不是通用代码模板所能自动正确推导的，一旦处理不当，即使代码在语法层面正确，也可能导致系统无法启动、外设异常或现场运行不稳定。

其次，嵌入式与 AIoT 系统通常具有明确的实时性要求。系统是否“基本可运行”并不等同于是否满足工程交付标准。实际应用中，还必须进一步考虑中断延迟、任务调度抖动、资源竞争、故障恢复路径、异常状态下的安全回退机制，以及长期连续运行时的稳定性。因此，“大多数情况下可运行”的代码，在工程实践中往往不足以支撑可维护、可验证、可部署的系统。

再次，AIoT 系统本质上是一个跨层协同系统，其正确性依赖于 MCU、RTOS、驱动、现场总线、边缘网关、遥测接口、OTA 升级机制以及云端数据模型之间的严格一致。无论是字段类型、单位换算、时间戳格式、状态机定义，还是协议报文结构，只要任一层出现错配，就可能引发链路中断、数据解释错误、控制失效，甚至造成远程升级失败和系统级异常。氛围编程通常更关注局部代码生成，而较少显式保证这种跨层一致性。

最后，代码能够编译通过仅表明其在语法和局部依赖层面是成立的，并不能替代真实工程环境下所需的系统验证工作。对于嵌入式与 AIoT 产品，仍必须经过硬件在环测试、协议互操作验证、故障注入测试、长期稳态运行评估、资源边界压力测试以及安全审查等多维验证流程，才能判断其是否真正具备工

程落地条件。换言之，编译成功并不等于系统可用，更不等于系统可信。

综合来看，“氛围编程”在嵌入式与 AIoT 场景中的核心局限，在于其更擅长生成“看起来合理的代码”，却难以天然保证硬件映射正确、时序约束满足、跨层定义一致以及系统验证充分。对于这类强约束、强耦合、强工程属性的系统开发，仅依赖提示驱动的代码生成方式往往是不够的，必须进一步引入面向嵌入式系统的 AI 辅助工程方法，通过架构约束、配置协同、验证流程与人工审查机制，将 AI 的生成能力纳入可控、可验证、可部署的工程框架之中。

4.2. 面向嵌入式系统的 AI 辅助工程

因此，“项目速度”并非将人工智能简单视为孤立的“自动代码生成器”，而是将其纳入一套以工作流驱动的系统工程框架之中，使 AI 参与嵌入式与 AIoT 系统开发的多个关键阶段，并始终受制于工程约束、验证要求与发布治理机制。该方法强调，AI 的作用不在于替代工程师完成最终设计决策，而在于辅助工程师提高需求理解、实现效率、验证覆盖度与文档一致性，从而支撑复杂系统从原型走向可部署工程实现，如表 2 所示。

Table 2. AI-assisted engineering process and guardrail mechanisms
表 2. AI 辅助工程流程与护栏机制

阶段	AI 支持内容	主要产出	约束与护栏
需求	拆解业务用例、定义接口规范、识别危险源	需求说明文档、接口草案、初步风险清单	人工复核系统边界与约束条件
实现	生成固件模板、协议封装代码、API 草稿	驱动框架、服务逻辑、数据模式定义	代码评审 + 静态分析检查
验证	生成单元测试用例、仿真场景、硬件在环(HIL)测试脚本	测试用例集、故障注入脚本、回归测试报告	持续集成(CI)回归测试 + 硬件在环验证
文档	整理设计说明、测试证据、标准运维程序(SOP)	系统设计文档、运维手册、版本发布说明	文档与测试证据留存，并与代码版本绑定
部署	生成监控规则、回滚预案、发布计划	发布实施方案、告警阈值配置、回滚操作方案	灰度发布策略 + 安全扫描审查

在需求分析阶段，AI 可用于辅助拆解业务用例、梳理系统边界、定义接口规范并识别潜在危险源，从而提高需求建模与早期风险识别的完整性。在实现阶段，AI 可加速生成固件模板、驱动初始化框架、协议封装代码、传感器数据模型描述以及边缘与云端接口草案，从而缩短从需求到初始实现的迭代周期。在验证阶段，AI 可进一步辅助生成单元测试用例、异常场景仿真脚本、协议一致性检查逻辑以及硬件在环测试脚本，以提高测试设计的系统性与覆盖范围。在文档阶段，AI 可用于自动整理设计说明、接口文档、测试证据与标准操作流程(SOP)，增强开发成果的可追溯性与知识复用能力。在部署阶段，AI 还可辅助生成发布说明、监控规则、告警策略及回滚预案，为后续运维治理提供结构化支持。

然而，效率的提升不能以牺牲系统可信性为代价。由于嵌入式与工业 AIoT 系统具有实时性强、软硬件耦合紧、现场环境复杂及故障代价高等特点，AI 参与开发流程时必须嵌入明确且强制性的质量护栏。具体而言，应在流程中纳入人工评审、持续集成(CI)回归测试、硬件在环验证、安全扫描以及受控发布机制，以确保 AI 生成内容不仅在语法层面成立，而且能够满足硬件约束、配置一致性、协议互通性与系统安全性的工程要求。

从工程方法论角度看，这种面向嵌入式系统的 AI 辅助工程，与“治理 - 映射 - 测量 - 管理”的风险

控制思想具有一致性，即 AI 不应被视为一次性产出的工具结果，而应被视为需要在系统全生命周期内持续约束、持续验证与持续治理的工程对象。尤其在工业场景中，任何由 AI 生成或辅助形成的设计与实现成果，最终都必须落实为可验证的接口规范、可测试的系统行为、可回退的版本管理机制以及可解释的运维操作指引。只有在这样的前提下，AI 才能真正从“生成代码的工具”转化为“提升嵌入式系统开发能力的工程助手”。

4.3. “三文件”提示策略

为解决 Zephyr 等复杂嵌入式项目中“AI 仅生成应用逻辑代码而忽略底层工程配置”的常见问题，本文提出一种面向嵌入式系统开发的“三文件”提示策略。其核心思想在于：针对任一具体功能需求，在提示设计阶段即明确要求 AI 同步生成.overlay/.dts、prj.conf 与 main.c 三类关键交付文件，从而使生成结果不再停留于孤立的业务逻辑层，而是覆盖硬件描述、系统配置与应用实现三个彼此耦合的工程层次。

其中，.overlay/.dts 文件用于完成硬件描述，主要定义设备树节点、引脚复用关系、总线类型与速率、外设地址及节点标签等内容，是应用程序正确访问底层硬件资源的基础；prj.conf 文件用于完成系统配置，负责显式启用 I2C、SPI、UART、BLE、日志、浮点运算、文件系统等功能所依赖的 Kconfig 选项，从而保证内核能力、驱动依赖与中间件组件在构建阶段被正确纳入；main.c 文件则承载应用层业务逻辑，负责调用系统 API、绑定设备树节点、组织任务流程，并实现状态机、控制逻辑及异常处理路径。三者分别对应硬件定义、软件配置与功能实现，共同构成一个可构建、可运行、可调试的完整嵌入式工程单元。

该策略的工程价值在于，它将抽象的“功能意图”直接映射为嵌入式工程落地所必需的三个关键构成层面，从而避免 AI 仅在逻辑层生成“看起来正确”的代码，却无法在真实工程中完成编译、链接与运行。以 Zephyr 为代表的现代嵌入式框架，其构建过程本质上是应用代码、Kconfig 配置与设备树描述三者的联合收敛过程。若任一环节缺失或不一致，均可能引发驱动未启用、设备节点不存在、头文件依赖缺失、编译选项不匹配，或运行时无法正确寻址外设等问题。因此，三文件同步交付不仅提升了 AI 生成内容的完整性，也显著增强了项目的可构建性与跨层一致性。

此外，该策略还有助于提升调试与问题定位效率。采用三文件同步交付后，一旦在构建或运行过程中出现异常，开发者可以首先判断问题来源究竟属于硬件描述层、系统配置层还是应用逻辑层，并据此进行有针对性的排查。这种分层式的问题定位方式，能够有效缩短调试路径，减少反复修改提示与盲目试错所带来的时间成本，从而提高 AI 辅助开发在复杂嵌入式工程中的实际可用性。

与“三文件”提示策略配套的另一关键机制是“上下文注入”。在提示过程中，必须显式向 AI 提供目标板卡型号、处理器架构、主时钟频率、所使用的传感器类型、总线接口方式、资源限制、实时性要求及安全边界等工程上下文信息，而不能让 AI 仅依据通用经验进行推断。对于深度嵌入式项目而言，越接近硬件层的实现细节，其对上下文完整性和准确性的依赖越强。只有在提示中提供充分、精确且与目标平台一致的上下文约束，才能最大程度减少“逻辑上合理但工程上不可用”的生成结果，并提高 AI 输出与真实系统需求之间的一致性。

总体而言，“三文件”提示策略并非简单增加 AI 的输出数量，而是通过显式约束其交付结构，使 AI 生成结果从“代码片段”提升为“工程单元”，如下图 7 所示。该策略结合上下文注入机制，为 AI 在 Zephyr 等嵌入式框架中的应用提供了一种更具工程可行性的方法路径，也为后续构建可验证、可维护、可部署的嵌入式 AI 辅助开发流程奠定了基础。

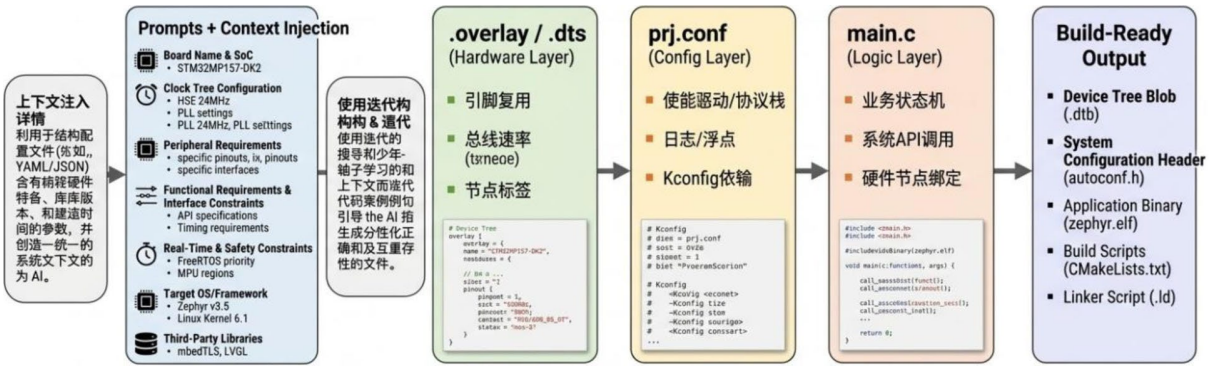


Figure 7. "Three-File" prompting strategy and context injection method
图 7. "三文件"提示策略与上下文注入方法

5. 实现栈与接口设计

在实现层面，系统采用三层架构进行技术栈划分：

MCU 层：主要由传感器驱动、RTOS 任务、信号特征提取模块与失效保护状态机构成。针对需快速本地响应的场景，可在该层执行轻量级阈值判断或 TinyML 推理，以规避总线与网络传输带来的不确定延迟[2] [3]。

边缘层：通常部署具备较强算力与接口扩展能力的工业网关或工控机，负责本地服务容器化、消息缓冲、设备管理、人机界面及多协议桥接。其中，OPC UA 为上层语义互操作提供统一信息建模框架[6]；MQTT 承担轻量遥测与远程指令传递[7]；而 CANopen/Modbus 则用于兼容现场设备与既有工业资产[8]。

云端层：承担对象存储、数据湖、模型注册、舰队看板及 CI/CD + OTA 管理等职能。与传统仅关注业务数据的云平台不同，本项目强调将模型版本、固件版本、配置版本与策略版本置于同一可追溯链条中，从而在发生异常时能清晰回答："当时运行的是哪个模型、哪版配置、在哪些节点上生效过"这一关键工程问题。为此，系统在横向能力上还需集成统一的身份与权限管理、全链路可观测性、策略控制及回滚机制。

从工程实践角度看，真正的难点不在于将单项技术接入系统，而在于确保各层接口在持续版本演进中保持一致性。例如，若现场节点上报的状态字节、边缘层推理结果的置信度字段、云端仪表板的解读信息以及运维人员接收的操作建议缺乏统一定义，将导致模型、软件与运维流程相互脱节。因此，接口规范、版本命名与证据留存机制应与代码一样，被视作系统设计的核心对象予以管理。

6. 负责任 AI 控制与应用示例

工业 AI 系统的构建并非简单地"接入一个模型"即可完成。与办公或消费类应用不同，工业场景尤其强调人在回路、安全覆盖、职责划分与异常处置的确定性。因此，"项目速度"将负责任 AI 的治理要求前置到系统设计阶段，而非事后补充。其核心控制原则包括：安全与人工干预机制、透明性与可解释性、数据最小化与隐私保护、问责与可追溯性，以及鲁棒性与网络安全。

在工程实现上，这些治理原则需要转化为可落地的控制机制，例如在控制链路中引入失效保护状态机与人工接管接口，在告警界面中展示异常原因码及建议检查路径，在更新流程中采用签名包、版本记录与灰度发布策略，并在运维侧保留审计日志、角色访问控制及远程策略变更记录。其核心目标是使 AI 能力始终运行于可验证、可约束、可回退的工程闭环之中。

以"项目速度"为例，该系统由 STM32 传感与执行节点、RK3588 边缘网关和云端 ThingsBoard 平台构成，形成 MCU－边缘－云协同闭环。在该架构中，AI 并不直接越过控制层驱动现场设备，而是通过

边缘策略判断、设备状态校验和末端状态机约束后才可能影响执行动作。例如，在环境调节场景中，即便上层平台发出加热或执行器控制建议，只要本地检测到温度越界、传感器异常、通信中断或人工切换到手动模式，STM32 节点均可拒绝执行该命令并进入预定义安全状态。该机制体现了“AI 可建议、系统可约束、人工可接管”的基本原则。

在透明性与可解释性方面，“项目速度”强调对异常路径和链路状态的显式展示。系统不仅呈现最终运行结果，还可展示关键传感量变化、节点在线状态、边缘转发情况、命令下发时间、执行确认状态及异常原因码。例如，对于“设备无响应”告警，系统可进一步区分为节点离线、总线超时或执行确认缺失等不同故障来源，从而提升故障定位效率，避免将异常简单归因于 AI 本身。

在数据治理方面，系统采用边缘优先处理、按需上云的策略。STM32 节点完成基础感知与局部控制闭环，RK3588 负责协议聚合、数据筛选和局部分析，仅将必要的遥测结果、事件信息和控制状态上传云端。该设计有助于降低带宽与存储压力，也为敏感数据本地处理和隐私保护提供了基础。若未来引入本地视觉模型或本地大模型能力，该分层架构同样支持“数据尽量留在边缘、云端同步结果与元数据”的治理思路。

在问责与可追溯性方面，系统通过完整链路记录实现控制行为的来源可查、过程可追和结果可验。一次控制动作可沿“云端发起 - 边缘转换 - 现场下发 - 节点执行 - 状态回传”的路径进行记录，为故障复盘、责任界定和合规审查提供依据。同时，系统通过本地降级运行、缓存与重传、受控升级与回滚等机制提升整体鲁棒性，避免网络或服务异常直接导致现场功能失控。

需要指出的是，负责任 AI 不仅体现在系统运行阶段，也体现在开发过程中的工程约束机制。“项目速度”所强调的 AI 辅助工程化方法(AI-assisted engineering)，并非让 AI 替代工程过程，而是将 AI 纳入需求分解、接口定义、配置生成、代码实现、测试验证和文档形成等环节，并通过人工审查、回归测试和硬件在环验证等方式对其输出进行约束。由此可见，工业 AIoT 系统中真正关键的并非单一模型性能，而是 AI 是否被置于一个可验证、可控制、可追责、可恢复的工程系统之中。

7. 评价维度与工程验证

由于本文工作的重点在于系统工程方法与组织流程，而非单一算法的精度评估，因此构建了一套涵盖四类维度的综合评价体系：

第一类：系统指标

包括端到端时延、离线可持续运行时间、OTA 回滚耗时等，用于衡量系统在现场环境下的整体可用性与故障恢复能力。

第二类：AI 性能指标

涵盖检测/分类准确率、误报率及模型漂移发现覆盖率，用于评估模型在实际场景中的有效性与稳定性。

第三类：负责任 AI 治理指标

包括告警解释覆盖率、事件可追溯性完整度、策略与权限违规次数等，用于考察系统在透明、可信、可审计方面的治理水平。

第四类：工程效率指标

涉及 AI 辅助节省的开发时间比例、自动生成的测试用例数量、发布后文档与代码的同步更新度等，用以体现 AI 辅助工程在组织与流程层面的实际收益。

该评价体系旨在从系统可靠性、AI 实用性、治理合规性与工程可持续性四个层面，综合反映工业 AIoT 项目在真实部署中的整体成效，如表 3 所示。

Table 3. Evaluation dimensions and representative metrics of Project Velocity
表 3. Project Velocity 评价维度与代表性指标

指标类别	代表性指标	说明
系统 KPI	端到端时延、离线可持续运行时间、OTA 回滚耗时	评估系统在现场环境下的整体可用性、网络中断时的连续运行能力，以及版本故障后的快速恢复能力
AI KPI	检测/分类准确率、误报率、模型漂移发现覆盖率	衡量模型在实际生产场景中的有效性与持续适应能力
负责任 AI KPI	告警解释覆盖率、事件可追溯完整度、权限/策略违规次数	考察系统在输出可解释性、治理闭环建立以及操作审计方面的成熟度与合规性
工程 KPI	AI 辅助节省时间比例、自动生成测试用例数量、文档与代码同步更新度	反映 AI 辅助工程流程对组织开发效率、交付质量一致性以及知识沉淀可持续性的实际贡献

需要强调的是，工业 AIoT 项目不应仅仅追求模型性能指标，而忽视整个工程链路中的综合成本与系统性风险。一个分类准确率更高、但无法实现安全回滚、缺乏结果解释能力或难以跨版本维护的系统，在实际生产环境中往往并不优于一个整体质量更均衡、可管控的解决方案。因此，“项目速度”将“可验证、可审计、可恢复”与传统的“精度、速度、效率”置于同等重要的地位。

“项目速度”目前已完成的是原型系统级验证，而非面向大规模量产场景的长期统计验证。尽管如此，该项目已经完成了若干具有明确工程意义的阶段性验收，表明所提出的方法并非停留在概念层面，而是已经在真实系统链路中得到验证。具体包括以下几个方面：

7.1. 端到端闭环链路验收已完成

系统已完成 STM32 传感节点、RK3588 边缘网关与 ThingsBoard 云平台之间的全链路打通，实现了从现场感知、边缘转发、云端可视化到控制命令下发、末端执行反馈的完整闭环。这表明本文提出的 MCU－边缘－云分层架构具有实际可运行性。

7.2. 协议桥接与跨层协同验收已完成

原型系统已验证现场节点到边缘侧、边缘侧到云端之间的数据转发与协议映射机制，证明异构节点、边缘平台和云端可在统一工程框架下协同工作。这一点对于工业 AIoT 系统尤为关键，因为实际难点往往不在单点模型，而在跨协议、跨设备、跨层级集成。

7.3. 离线退化运行能力已完成原型验证

在云端不可达或链路异常情况下，系统仍可由现场节点和边缘网关维持基本控制逻辑，验证了“边云解耦、现场优先、安全退化”的设计原则。该结果表明系统具备一定的鲁棒性基础，而非完全依赖云端在线运行。

7.4. 可追溯与治理链路已完成原型验证

对控制命令的来源、边缘转发过程、节点执行状态与结果回传路径，系统已具备基本日志记录与事件追踪能力，可支持故障定位和责任追查。这说明本文提出的负责任 AI 治理要求已在工程实现中具有初步落点。

7.5. AI 辅助工程流程已完成可行性验证

“项目速度”采用 AI-assisted engineering 方法完成需求分解、接口定义、配置生成、部分代码实现、

测试辅助和文档整理，显著缩短了从概念到原型系统落地的周期。该结果验证了本文所强调的重点：AI在工业 AIoT 中的价值不仅在于模型推理，更在于提升跨层工程协同效率。

需要强调的是，本文当前提供的验证结果属于原型级、方法级和链路级验收，其目标是证明所提方法“能够工作、能够集成、能够闭环、能够治理”，而非宣称已完成面向大规模商用部署的最终性能定型。因此，本文并未将有限场景下的单次实验结果简单外推为通用结论，而是将其定位为一种经过原型系统验证的工程方法与架构路径。

从工程研究的角度看，工业 AIoT 项目不应仅追求单一模型性能指标，而忽视系统链路中的综合成本与系统性风险。一个即使在局部分类任务上精度更高、但缺乏安全回滚、异常解释、权限约束和版本维护能力的系统，在真实生产环境中并不一定优于一个整体质量更均衡、可验证、可审计、可恢复的解决方案。因此，“项目速度”将“可验证、可审计、可恢复”与传统“精度、速度、效率”置于同等重要的位置。

目前，该方法已在“项目速度”演示系统中完成了整体架构、端到端数据链路、协议桥接、边云协同与工程流程的原型验证，并为后续的硬件在环测试、故障注入测试、长期稳定性测试及更大规模舰队级验证预留了接口。后续研究将重点关注以下方向：

1) 补充更系统的量化实验数据，如端到端时延分布、离线运行持续时间、回滚恢复时间与人工接管成功率；

2) 建立面向设备侧模型漂移的持续监测机制；

3) 完善审计轨迹记录与权限治理策略；

4) 开展跨多场景、长周期的稳定性评估与对比实验。

8. 结论

本文围绕“项目速度”原型系统，提出了一套面向工业 AIoT 复杂系统的 AI 辅助工程方法，系统阐述了包括混合 MCU－边缘－云架构、端到端决策闭环、“三文件”提示策略、负责任 AI 控制机制以及多维评价框架在内的关键技术要素。与单纯依赖对话式代码生成不同，本文强调将 AI 工具整合进受多重质量护栏约束的系统工程流程中，使其在提升开发效率的同时，不破坏嵌入式系统对时序确定性、功能安全与全链路可追溯性的基本要求。

从工程实践角度看，工业 AIoT 系统的关键并非追求某一层的孤立“智能”，而在于实现各层级在职责划分、接口契约、版本治理与故障恢复上的协同一致性。只有当系统能够在本地实现实时、可靠的闭环控制，在边缘侧完成上下文感知与即时响应，在云端实现集中治理与持续学习，并且整个链路始终保持可解释、可回滚与可审计，AI 辅助工程才真正具备在工业场景中规模化落地的价值。

需要说明的是，“项目速度”的作用在于完成原型级系统验证与链路验收，而非形成某一单任务上的算法竞赛结果。后续我们将结合“项目速度”，作为一个基于 AI 辅助工程的混合 MCU－边缘－云平台的工业 AIoT 系统的开源项目，详细描述硬件的实现，边缘 RK3588 的 AI 算法的实现，本地 LLM 的集成，以及各个行业的应用，包括具身智能。大家可以通过[9]获取最新的进展。

参考文献

[1] Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016) Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, **3**, 637-646. https://doi.org/10.1109/jiot.2016.2579198

[2] 吴薇, 吕亚欣. 服务国外市场的 AIoT 开源方案及其应用[J]. 单片机与嵌入式系统应用, 2022, 22(9): 4-9.

[3] Warden, P. and Situnayake, D. (2020) TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. O’Reilly Media.

[4] Lin, J., Chen, W.-M., Lin, Y., *et al*. (2020) MCUNet: Tiny Deep Learning on IoT Devices. arXiv:2007.10319.
[5] The Zephyr Project (2026) Zephyr Project Documentation. https://docs.zephyrproject.org/latest/index.html
[6] OPC Foundation (2026) OPC Unified Architecture (OPC UA), IEC 62541 Overview.
[7] OASIS Open (2019) MQTT Version 5.0.
[8] CAN in Automation (CiA) (n.d.) CANopen CC—The Standardized Embedded Network.
[9] http://www.cynoware.com/cynoware/

Call for Papers

Embedded Technology and Intelligent Systems

嵌入式技术与智能系统

国际中文期刊征文启事

https://www.hanspub.org/journal/etis

ISSN: 3065-1220

《嵌入式技术与智能系统》是一本开放获取、关注集成传统嵌入式技术与新兴智能系统的前沿研究最新进展的国际中文期刊，期刊特别注重软件算法、芯片设计与硬件实施的协同进展，以及理论研究与工程实践的紧密结合，面向学术界学者、产业界专家与工程师、学生及技术爱好者，关注中国领先产业集群的广阔发展潜力。本期刊强调发表原创性、创新性及具有实用价值的研究成果。 该期刊由汉斯出版社出版，全球发行，现诚邀相关领域的学者投稿。

投稿领域：

人工智能技术-边缘计算-端侧智能和大模型嵌入式应用
GPT-行业GPT以及GPT在嵌入式及智能系统研发中的应用
信息物理融合系统(CPS)-物联网技术-感知计算和无线传感网-泛在电力物联网-智能电表-储能技术-智能输变电
嵌入式系统结构-嵌入式操作系统与中间件-Linux、安卓和开源鸿蒙应用
实时操作系统-虚拟化和容器技术-混合关键系统
嵌入式软件形式化建模-软件测试和仿真-功能安全技术
嵌入式软件云原生技术-CI/CD和DevOpt-微服务
软硬件协同设计-开源指令集和开源芯片-RISC-V产业生态
嵌入式SoC技术--MCU 创新与生态-FPGA/DSP技术和应用
AI芯片和算法-存储技术-GPU 技术-视觉芯片及嵌入式显控应用
CAN和工业总线技术-时间敏感系统-电机控制-PLC和工业PC
无线通信技术-WiFi/蓝牙/Mesh/蜂窝/5G网络-物联网安全-低功耗设计
嵌入式系统课程改革-物联网和AI教学研究-职业教育-企业人才培养
嵌入式智能系统应用（智能家居、可穿戴设备、机器人、医疗电子、汽车电子和航空航天等）

征文要求及注意事项：

1. 稿件务求主题新颖、论点明确、论据可靠、数字准确、文字精炼、逻辑严谨、文字通顺，具有科学性、先进性和实用性；
2. 稿件必须为中文,且须加有英文标题、作者信息、摘要、关键词和规范的参考文献列表；
3. 稿件请采用WORD排版，包括所有的文字、表格、图表、附注及参考文献；
4. 从稿件成功投递之日起，在2个月内请勿重复投递至其他刊物。本刊不发表已公开发表过的论文。文章严禁抄袭，否则后果自负；
5. 本刊采用同行评审的方式，审稿周期一般为5~14日。

欲了解更多信息请登录 https://www.hanspub.org/journal/etis　　联系邮箱：etis@hanspub.org

Hans汉斯

嵌入式技术与智能系统

主编：何立民　北京航空航天大学教授

主办：汉斯出版社　珠海昊谷电子科技有限公司

编辑：《嵌入式技术与智能系统》编委会

网址：https://www.hanspub.org/journal/etis

电子邮箱：etis@hanspub.org

出版：汉斯出版社